ABSTRACT

MATHEMATICS

WATTS, MARVIN DWAYNE B.S. CLARK ATLANTA UNIVERSITY, 1998

TRANSIENT SOLUTION OF COMPRESSIBLE VISCOUS FLOWS ON HIGH PERFORMANCE COMPUTING PLATFORMS USING FINITE VOLUME METHODS

Research Advisor: Professor Shahrouz Aliabadi

Committee Chairman: Professor Khalil Shujaee
Dissertation dated December 2006

The work presented in this dissertation is the result of "application-driven" research, with the need to solve complex large-scale engineering problems of significance and relevance to the Army and NASA using state-of-the-art high performance computing (HPC) platforms as its primary motivation. Currently, a majority of commercially available computational fluid dynamics (CFD) simulation algorithms in use by Army and NASA researchers and scientists solve the Navier-Stokes equations using a finite volume method (FVM) framework. Although these codes are extremely mature and take advantage of the numerical schemes complimentary to FVM, many do lack in computational performance for second-order accurate time integration schemes, due to the resulting nonlinear system of equations for large-scale applications, and exhibit poor scalability on a number of supercomputing platforms.

Therefore, the purpose of this work is the development of a fully implicit, finite volume solver for large-scale transient compressible viscous flows, optimized for implementation on parallel, vector, and multi-streaming architectures. Optimization will include reduction in memory requirements, increasing computation speed, and obtaining near-linear code scalability. This is accomplished through implementation of innovative Jacobian-free/matrix-free iterative algorithms and code parallelization and vectorization. The Jacobian-free Generalized Minimal RESidual (GMRES) method is used to solve the resulting linear system inside each nonlinear Newton-Raphson iteration. Furthermore, the matrix-free Lower-Upper Symmetric Gauss Seidel (LU-SGS) method is employed as a preconditioning technique to the GMRES solver.

Massively parallel implicit computations of both 2-dimensional and 3-dimensional aerodynamic applications using vector/multi-streaming and cluster supercomputers are presented to demonstrate the performance of the present solver in several aspects. These applications show the current implementation to be highly robust and accurate for problems of all flow regimes, subsonic, transonic, and supersonic. Though not originally intended for subsonic flows within the incompressible limit, i.e. flows with Mach numbers of 0.3 or less, results are presented which show that the solution accuracy of this solver is maintained for this class of problem. However, additional cases would need to be studied to determine the full scope of application to subsonic flows. The scalability of the current implementation is shown to be near-linear and super-linear across multiple supercomputing platforms.

TRANSIENT SOLUTION OF COMPRESSIBLE VISCOUS FLOWS ON HIGH PERFORMANCE COMPUTING PLATFORMS USING FINITE VOLUME METHODS

A DISSERTATION

SUBMITTED TO THE FACULTY OF CLARK ATLANTA UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF DOCTOR OF SYSTEMS SCIENCE

BY

MARVIN D. WATTS

DEPARTMENT OF ENGINEERING

ATLANTA, GEORGIA

MAY 2007

ACKNOWLEDGEMENTS

My sincere thanks go to my advisor, Shahrouz Aliabadi, for six years of support and guidance through both my undergraduate and graduate matriculation and Dr. Shuangzhang (Shane) Tu, who was always a pleasure to work with. I also wish to thank Professors Khalil Shujaee, Larry Wang, Roy George and Mr. Charles Nietubicz who devoted time out of their busy schedules to serve on my defense committee. Finally, this work is funded in part by the Army High Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory contract numbers DAAD 19-01-2-0014 and DAAD 19-03-D-0001, and the National Science Foundation. Partial support for this publication was made possible through the Northrop Grumman Center for High Performance Computing of Ship Systems Engineering at Jackson State University.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In the area of computational fluid dynamics (CFD), the engineering problems investigated can be categorized under two very broad headings, compressible flows or incompressible flows. The division of problems into one of these two categories is governed by the fluctuation of density within a given flow field. Specifically, the rule of thumb is that a flow can be assumed to be incompressible when the Mach number, defined in Chapter 2, is less than 0.3. The justification for this assumption lies in the fact that the variation of density can be shown to be less than 5% when the Mach number is less than 0.3 [1]. In reality all flows are compressible, but the assumption of incompressibility allows the density to be treated as a constant. When the density is constant, or at least assumed to be constant, a simplification can be made to the compressible Navier-Stokes equations, which are the set of equations describing the motion of fluids; conservation of mass, conservation of momentum, and conservation of energy; leading to the derivation of a set of equations referred to as the incompressible Navier-Stokes equations [1]. These incompressible Navier-Stokes equations consist of a zero-divergence or continuity condition, which characterizes the mass conservation. Furthermore, a decoupling occurs between the conservation of energy and the conservation of momentum equations. As such, the system of equations describing incompressible flow requires only the continuity equation and conservation of

momentum equation, in a strict sense. If temperature effects within the flow are desired, the conservation of energy equation must then be included. Due to the inherent assumption used to derive the incompressible Navier-Stokes equations, applications are limited to a much smaller, though still practical and important, sub-set of engineering problems, as compared to the set of engineering problems within the category of compressible flows. Therefore, emphasis is placed upon development of an appropriate numerical scheme for the compressible Navier-Stokes equations, which are presented in more detail in Chapter 2.

1.1 Numerical Methods for Solving the Navier-Stokes Equations

The selection of an appropriate numerical method for the solution of partial differential equations (PDE), specifically the compressible Navier-Stokes equations on unstructured meshes, is crucial to the development of robust and accurate algorithms for practical engineering problems. Currently, two numerical methods dominate the field of CFD. Those methods are the stabilized finite element method (FEM) and the finite volume method (FVM).

1.1.1 Finite Element Methods

The finite element method is based upon the Galerkin approximation method [2]. The standard Galerkin method works excellently for diffusion-dominated problems. This is because the standard Galerkin method is inherently a central differencing method. For advection-dominated flows, however, the classical Galerkin method must be stabilized to remove nonphysical oscillations. The main idea behind stabilization is to add extra mesh-dependent terms to the standard Galerkin method. The terms are designed to maintain the solution consistency between the solution to the modified differential equation and the

original differential equation. The extra terms may be derived by modifying the weighting function.

There are many stabilization procedures to obtain the identical stabilized form [3, 4]. These approaches include the Streamline-Upwind/Petrov-Galerkin (SUPG) [4, 5] methods, Galerkin Least Square (GLS) [6] and Finite Increment Calculus (FIC) [7]. The SUPG idea comes from the analogy between adding balancing diffusion and upwinding, i.e. adding the necessary stabilization for advection-dominant flows by considering the direction of propagation of information. The convection is only active in the direction of the resultant fluid velocity, and thus the diffusion introduced by upwinding should be anisotropic. The SUPG method adds extra diffusion in the direction of the streamlines and avoids excess crosswind diffusion. The GLS method adopts the idea of lease square residual minimization and is equivalent to the SUPG method when the piecewise linear approximations are used. The FIC method re-derives the original conservation laws on a finite size domain instead of an infinitesimal spatial-time domain. The derivation is based on higher order Taylor expansions. The resulting formulation is similar to that of the SUPG method. The stabilized finite element method has been used by many authors to solve both incompressible and compressible fluid dynamics problems [4-19].

The main disadvantage of the stabilized finite element method is the difficulty in determining appropriate stabilization parameters. The stabilization parameters in the SUPG stabilized finite element method rely heavily upon the definition of the characteristic element length. For isotropic unstructured meshes, the characteristic length is well defined. However for high Reynolds number viscous flow, where high aspect ratio elements are used inside boundary layer, the element length is not well defined. An

inappropriate amount of stabilization, excessive or insufficient, can lead to a loss of accuracy inside the boundary layer.

1.1.2 Finite Volume Methods

The finite volume method works excellently for advection-dominated problems. This is because the finite volume method makes use of upwinding methods to discretize hyperbolic equations according to the direction of wave propagation [20]. These methods include exact or approximate Riemann solutions or flux vector splitting (FVS) techniques to evaluate the inviscid flux term. The exact Riemann solution for a general $n \times n$ non-linear hyperbolic system consists of $n+1$ constant states separated by n waves. For each eigenvalue there is a wave family. Due to the nonlinearity of the exact Riemann problem, the exact solution may only be computed by numerical iteration, which requires considerable computational effort. As such, an approximate solution is typically computed. A more detailed discussion of the nature of Riemann solutions can be found in [21]. FVS techniques use the sign of eigenvalues of the coefficient matrix to decompose the flux vector into its positive and negative components. Though more diffusive than it's exact or approximate Riemann solution counterpart, the determination of upwind directions is done with less computational effort [22]. Also, the added diffusion eliminates the possibility of spurious oscillations when computing grid-aligned shock waves, as is the case for blunt body flow problems at supersonic speeds [23, 24].

Due to the many choices of approximate Riemann solvers and flux vector splitting schemes, such as the Harten, Lax and van Leer (HLL) family of flux functions [21, 25-27], the Modified Steger-Warming flux function [28], the Marquina flux function [29], the Local Lax-Friedrich flux function [30], Roe's method [31], etc., a virtual library of

inviscid flux solvers can be implemented into any FVM algorithm, thus ensuring robustness and applicability to a wide range of practical engineering problems. However, nonphysical oscillations can still appear near flow discontinuities during the solution reconstruction process. To suppress these overshoots and undershoots, a slope limiting procedure is often used [32].

1.2 Turbulence Modeling

The lift and drag contributions for compressible viscous flows are highly dependent upon flow separation. Flow separation refers to the phenomena of reversed-flow, which is a consequence of both friction and adverse pressure gradients within the boundary layer [1]. It is caused by the complete change in direction of fluid elements originally moving upstream to downstream near the surface of solid bodies suddenly moving back upstream. The overall result of this flow separation is the production of drag and considerable loss of lift.

Turbulence is an effective means of stalling flow separation. Within turbulent flows, there is a range of scales of fluid motion. The fluid elements close to the surface have more energy, which prevents them from readily succumbing to the effects of adverse pressure gradients present in the boundary layer. Furthermore, the separated region and resulting wake field behind the solid body will be smaller, thus reducing the pressure drag due to separation. However, the benefits of turbulence are influenced by the body shape. In general, turbulent flow is most desirable for blunt bodies, but exceptions can exist.

The Reynolds number, defined in Chapter 2, plays an important role in determining the influence of turbulence in viscous flows. The Reynolds number is the ratio of inertial

forces to viscous forces, and as such can reveal the dominant forces for a specific flow. Details of the Reynolds number and its relevance to turbulence are given below.

At this point it is clear that in addition to selecting an appropriate numerical method, identifying a proper turbulence model is equally as important to the overall effectiveness of any solver applied to compressible viscous flows. For the purposes of numerical simulation, several types of turbulence models, each with specific advantages and disadvantages, are currently in use. The Large Eddy Simulation (LES), the Reynolds-Averaged Navier-Stokes (RANS) equations, and the Detached Eddy Simulation (DES) turbulence models currently serve as the primary choices for accurate turbulence modeling.

1.2.1 Direct Numerical Simulation

The ideal approach to turbulent simulation is the direct resolution of all relevant scales of motion using only the discretized Navier-Stokes equations, which is referred to as Direct Numerical Simulation (DNS). However, the range of relevant scales of motion in any flow increase with the Reynolds number, R_e. The length scale of the smallest scales of motion can be estimated to be on the order of $O\left(R_e^{-3/4}\right)$, according Kolmogorov's theory [33]. Therefore, the resolution of all relevant scales of motion in all three dimensions would require a grid on the order of $O\left(R_e^{9/4}\right)$. Considering that the Reynolds number for most practical engineering problems is on the order of $R_e \approx 10^6$, the number of required grid cells would easily make such calculations impractical using current computational resources.

1.2.2 Large Eddy Simulation

Large Eddy Simulation, as the name implies, focuses on the resolution of the largest or mean scales of motion, or eddies, through spatial filtering, and the effect of the smaller scales upon the resolved scales is modeled using some type of subgrid-scale (SGS) model, such as the Smagorinski eddy-viscosity model, which takes advantage of the isotropic nature of turbulence [34-36]. In this way, the mechanism of energy dissipation from the largest scales to the smallest scales is properly modeled. Unfortunately, LES does require that the computational grid be sufficiently fine to resolve the largest eddies, but not so fine as to resolve the full spectrum of relevant scales of motion. Due to this requirement, the computational grid size for certain problems can be comparable to that of DNS. This requirement is further compounded by the need for additional cells to resolve relevant scales when the turbulence is anisotropic, as in near-wall and wall-bounded flows.

1.2.3 Reynolds-Averaged Navier-Stokes Simulation

Conversely, the Reynolds-Averaged Navier-Stokes equations are derived through decomposition of the total solution into a mean, or average, of the solution and superimposed fluctuations, where the fluctuations reflect the turbulent intensities of the variables [34, 37]. The effect of the fluctuations on the average solution are modeled using some type of eddy-viscosity model, which typically applies temporal averaging to the turbulent flow, thus reducing it to a simple steady state laminar flow. This drastically reduces the computational grid size requirements when compared with DNS and LES methods, making RANS an inexpensive method of turbulence modeling. RANS gives

good results in near-wall and wall-bounded flows, but generally fails for massively separated flows.

1.2.4 Detached Eddy Simulation

It is evident from the discussion above that, in general, areas in which LES has difficulty, RANS excels, whereas the short comings of RANS are not seen in LES. As such, Detached Eddy Simulation (DES) was developed as a combination of these methods. In its intended use, the boundary layers near the wall surfaces are treated using a RANS model and the regions of separated flow away from the wall are modeled using LES [38, 39]. The wall-parallel grid spacing is used as a 'switch' between these two models. Though relatively new, this method has already been applied by many authors to the solution of turbulent flows [40-42].

1.3 Industry Standards

In the arena of commercial software developed for the solution of CFD problems, FVM is the most widely used numerical method by CFD researchers to solve the compressible Navier-Stokes equations. Such codes as *CFD++*, *FLUENT*, *Star-CD*, and a list of other flow solvers all solve the Navier-Stokes equations in a finite volume framework. In addition, these codes possess multiple turbulence models, approximate Riemann solvers, and time integration schemes; any combination of which can be selected by the user. Although these codes are extremely mature and take advantage of the numerical schemes complimentary to FVM, many do lack in computational performance for second-order accurate time integration schemes, due to the resulting large nonlinear system of equations for large-scale applications, and exhibit poor scalability on a number of supercomputing platforms. The performance of several

commercial FVM codes was studied on a Linux Network Cluster, the results of which are summarized in [43]. Interestingly, many of the codes examined exhibit linear scalability up to only 64 processors, and some less. Given that supercomputing platforms with as many as 4000 processors are currently in development for release as early as 2007, this lack of scalability degrades the advantage of using these state-of-the-art platforms.

1.4 Research Objective

Therefore, the purpose of this work is the development of a fully implicit, finite volume solver for compressible viscous flows common to external ballistics, optimized for implementation on parallel, vector, and multi-streaming architectures. Optimization will include reduction in memory requirements, increasing computation speed, and obtaining near-linear code scalability.

Implicit time integration is adopted to obtain better efficiency, especially for high Reynolds number flows. Usually, the Newton-Raphson iterative method is used to solve this nonlinear system. Inside each Newton-Raphson iteration, a sparse and often ill-conditioned linear system must be solved. Since the last decade, the Generalized Minimal RESidual (GMRES) solver introduced by Saad [44] has been widely used in solving large sparse linear systems. The beauty of the GMRES solver lies in its excellent convergence characteristics. In addition, the GMRES solver involves only matrix-vector multiplication, thus a Jacobian-free implementation is possible. With a Jacobian-free approach, we do not have to store the huge sparse Jacobian matrix and the memory requirement for large applications can be significantly reduced. Like other iterative methods, the performance of the GMRES solver is highly related to the preconditioning

technique. Knoll and Keyes [45] have made an extensive survey of the Jacobian-free Newton-Krylov (JFNK) method where various preconditioning techniques have been reviewed. Though the GMRES solver itself can be Jacobian-free (matrix free), the preconditioning technique is usually not matrix-free. Luo et al. [46] and Sharov et al. [47] proposed a matrix-free preconditioning approach for the GMRES solver. In their approach, the Jacobian obtained from the low-order dissipative flux function is used to precondition the Jacobian matrix obtained from higher-order flux functions. Using the approximate Lower Upper-Symmetric Gauss-Seidel (LU-SGS) factorization of the preconditioning matrix, their preconditioning is truly matrix-free. With the Jacobian-free GMRES solver and the matrix-free LU-SGS preconditioning, the majority of the memory consumed by the FVM code depends solely on the size of the Krylov space in the GMRES solver. The combined LU-SGS/GMRES method in Refs. [46, 47] was developed for shared-memory platforms. Here, we extend it for distributed-memory platforms using the mesh partitioning technique and the Message Passing Interface (MPI) functions.

Parallel computing using multiple processors simultaneously makes the solution of larger problems attainable in less time. If the parallel computer is also equipped with multi-streaming and vector processors, the process of simulating and analyzing large-scale problems will be further accelerated.

The Cray X1E is such a supercomputer built using a hybrid parallel, vector, and multi-streaming design. The basic component of the Cray X1E is the Multi-Streaming Processor (MSP), which is a multi-module chip composed of four Single-Streaming Processor (SSP) modules and four cache modules. The MSP is the user-addressable

processing element, and the division of work (i.e. streaming) of the four SSP modules is directed by the compiler (however, the user does have the ability to dictate the streaming process by using Cray Streaming Directives). Each SSP module is a fast vector processor, which contains 2 vector pipes and 32 vector registers. Each SSP also contains a slower scalar processing element, and because of this scalar performance slowness, codes running on the Cray X1E should be fully vectorized. Any significant scalar calculations in the code will be an overall drag on performance and limit the computer's overall effectiveness.

Furthermore, due to the recent growth in Linux cluster computing power, the Linux Network Evolocity II cluster is also a viable resource for solving large-scale problems. This computer is comprised of five different nodes: compute, analysis, storage, login, and management. All nodes contain dual 3.6 GHz Intel XEON EM64T processors. The 1024 compute nodes have 4GB memory each, totaling 2048 processors and 4 TB of memory with a peak system performance of 14.7 Teraflops.

Two main subroutines need to be vectorized. Both subroutines are called extensively by the Jacobian-free GMRES solver and consume the vast majority of the total computational time. Therefore, the vectorization of these two subroutines plays a crucial role in speeding up the code. One subroutine is a long loop over faces. Inside the loop, the inviscid and viscous fluxes across each face are computed and distributed to the face's two adjacent cells. The residual vector for each cell can thus be assembled. In our approach, we separate all faces into groups according to the so-called "face coloring" algorithm [48]. Inside each group, no two faces share the same cell neighbors and the loop can be forced to vectorize without worrying about accessing the same cell at the

same time by two faces, thus avoiding possible memory conflicts. Another subroutine is the LU-SGS preconditioning. Because this preconditioning is based on the LU approximate factorization, it involves solving two block triangular linear equation systems. However, exact solution of triangular systems requires sequential computations, which makes the vectorization impossible. To avoid the sequential computations in solving the triangular system and make vectorization possible, we apply the approximate truncated Neumann expansions of the inverse of the triangular matrices [49]. Numerical experience shows that this approximation still yields very good preconditioning performance in helping the convergence of the GMRES solver.

We also present our implementation of a simple slope limiting technique based on Local Extremum Diminishing (LED) principle. The limiting procedure ensures that no new extrema are allowed during reconstruction. This limiter effectively suppresses the unphysical overshoots/undershoots while not hampering the convergence. Also, through an adjustable parameter, the limiting amount can be controlled by the user.

1.5 Dissertation Structure

The remainder of this dissertation is organized as follows. In Chapter 2, we present the governing equations of fluid mechanics and heat transfer. Additionally, a brief description of the current turbulence model, which is used to compute the turbulent eddy viscosity, is given. Chapter 3 details the finite volume spatial discretization, time integration, linear system solver and preconditioning technique. Chapter 4 presents our implementation of linear reconstruction, with details about slope limiting. Chapter 4 also gives the formula to compute the gradient at the face in discretizing the viscous terms. Our turbulence model implementation is discussed in Chapter 5. Chapter 6 discusses the

parallelization and vectorization of the code on distributed memory supercomputers. In Chapter 7, numerical examples are presented to illustrate the application of these methods. Finally, we summarize with conclusions in Chapter 8.

CHAPTER 2

PROBLEM STATEMENT

In this chapter, the equations which govern the mechanics of fluid and heat transfer are reviewed. We start with the fundamental equations of fluid dynamics which are based on conservation of mass, momentum, and energy. These equations are expressed in the compact vector form which provides a convenient environment for our finite volume formulation. Also, our current turbulence model is presented. Finally, we briefly describe the non-dimensional form of the equations for compressible flows.

2.1 Governing Equations

Our current finite volume solver has been developed for solving the unsteady compressible Navier-Stokes equations in the conservative form;

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_j}{\partial x_j} - \frac{\partial \mathbf{E}_j}{\partial x_j} = 0 \text{ on domain } (\mathbf{x},\ t) \in \Omega \times (0,\ T). \tag{2.1}$$

where the repeated indices represent summation convention. Here Ω is the triangulated spatial domain with boundary $\partial\Omega$ and $(0, T)$ the time domain. An initial condition, $\mathbf{U}_0(\mathbf{x}) = \mathbf{U}(\mathbf{x},\ t = 0)$ and appropriate boundary conditions on $\partial\Omega$ must also be given. $\mathbf{U}$, $\mathbf{F}_j$ and $\mathbf{E}_j$ represent the conservative state vector, the inviscid flux vector and the viscous flux vector, respectively. They are defined as

$$\mathbf{U}=\begin{bmatrix}\rho\\ \rho u_i\\ \rho E\end{bmatrix},\ \mathbf{F}_j=\begin{bmatrix}\rho u_j\\ \rho u_i u_j+\delta_{ij}p\\ (\rho E+p)u_j\end{bmatrix},\ \mathbf{E}_j=\begin{bmatrix}0\\ \tau_{ji}\\ -q_j+\tau_{jk}u_k\end{bmatrix}. \tag{2.2}$$

where ρ, u_j, E, p, q_j and τ_{ji} are density, the components of the velocity, the specific total energy, pressure, heat flux, and viscous stress tensor, respectively. The shear stress tensor and the heat flux vector are defined as;

$$\tau_{ji}=(\mu+\mu_t)\left[\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)-\frac{2}{3}\left(\frac{\partial u_i}{\partial x_i}\right)\delta_{ij}\right], \text{ and} \tag{2.3}$$

$$q_j=-(k+k_t)\frac{\partial T}{\partial x_j}, \tag{2.4}$$

where μ, μ_t, k, k_t and T are laminar viscosity, turbulent viscosity, laminar thermal conductivity, turbulent thermal conductivity and temperature, respectively.

2.2 Turbulence Model

As stated in the Introduction, both Large Eddy Simulation (LES) [34-36] and the Reynolds Averaged Navier-Stokes (RANS) [34, 37] turbulence models perform well in different situations. As such, Detached Eddy Simulation (DES) [38, 39] was developed as a combination of these two methods. In its intended use, the boundary layers near the wall surfaces are treated using a RANS model and the regions of separated flow away from the wall are modeled using LES. The wall-parallel grid spacing is used as a 'switch' between these two models. In the rest of this section, we present the DES formulation.

Currently, we use a Spalart-Allmaras (SA) one-equation DES turbulence model to provide the turbulent eddy viscosity [38-42];

$$\frac{D\tilde{\nu}}{Dt} = c_{b1}S\tilde{\nu} - c_{w1}f_w\left[\frac{\tilde{\nu}}{\tilde{d}}\right]^2 + \frac{1}{\sigma}\left[\nabla\cdot\left((\nu+\tilde{\nu})\nabla\tilde{\nu}\right) + c_{b2}(\nabla\tilde{\nu})^2\right]. \tag{2.5}$$

The turbulent eddy viscosity is obtained from;

$$\nu_t = \tilde{\nu} f_{v1}, \tag{2.6a}$$

where

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}, \tag{2.6b}$$

$$\chi \equiv \frac{\tilde{\nu}}{\nu}, \tag{2.6c}$$

and ν is the molecular viscosity.

$$\tilde{d} \equiv \min(d, \mathbf{C_{DES}}\Delta), \tag{2.7a}$$

where

$$\Delta \equiv \max(\Delta x, \Delta y, \Delta z). \tag{2.7b}$$

d is the distance to the nearest solid wall and Δx, Δy, and Δz are the grid spacing. The model constant $\mathbf{C_{DES}}$ is of the order one, and based on experiment is typically set to 0.65. S is the magnitude of the vorticity. The function f_w is given by;

$$f_w = g\left[\frac{1+c_{w3}^6}{g^6+c_{w3}^6}\right]^{1/6}, \tag{2.8a}$$

where

$$g = r + c_{w2}\left(r^6 - r\right), \text{ and} \tag{2.8b}$$

$$r = \frac{\tilde{v}}{Sk^2\tilde{d}^2}. \tag{2.8c}$$

The remaining constants are $c_{b1} = 0.1355$, $\sigma = 2/3$, $c_{b2} = 0.622$, $k = 0.41$,

$c_{w1} = \frac{c_{b1}}{k^2} + \frac{(1+c_{b2})}{\sigma}$, $c_{w2} = 0.3$, $c_{w3} = 2$, $c_{v1} = 7.1$.

Ideally, $\tilde{d} = d$ corresponds to the near-wall region of the flow field, and consequently the S-A RANS feature of this formulation is active. When the flow is near equilibrium, the production/destruction terms of the model are balanced;

$$c_{b1}S\tilde{v} \approx c_{w1}f_w\left[\frac{\tilde{v}}{\tilde{d}}\right]^2, \tag{2.9a}$$

or

$$\tilde{v} \approx coefficient * \tilde{d}^2 S. \tag{2.9b}$$

Thus, $\tilde{d} = \mathbf{C_{DES}}\Delta$ in the detached flow, or LES, region yields a Smagorinsky eddy viscosity.

2.3 Non-dimensional Form of Equations

Not all the terms in the governing equations have the same importance. This can not be observed by looking at the dimensional equations. Casting the governing fluid dynamics equations in non-dimensional form results in working with non-dimensional parameters such as the Mach number, Reynolds number, Prandtl number, etc. These non-dimensional quantities are independent of each other and their values yield sufficient information about imposing the correct boundary conditions and selecting the right

computational domain. There are many different non-dimensionalizing procedures. The non-dimensional quantities we use are as follows;

$$\mathbf{x}^* = \frac{\mathbf{x}}{L}, \qquad t^* = \frac{\|\mathbf{u}_\infty\| t}{L}, \qquad \mathbf{u}^* = \frac{\mathbf{u}}{\|\mathbf{u}_\infty\|},$$

$$\rho^* = \frac{\rho}{\rho_\infty}, \qquad p^* = \frac{p}{\rho_\infty \|\mathbf{u}_\infty\|^2}, \qquad i^* = \frac{i}{\|\mathbf{u}_\infty\|^2},$$

$$\mu^* = \frac{\mu}{R_e \mu_\infty}, \qquad \lambda^* = -\frac{2}{3}\mu^*, \qquad \left(\frac{\kappa}{R}\right)^* = \frac{\gamma \mu^*}{(\gamma - 1) P_r},$$

where the non-dimensional variables are denoted by an asterisk. Here the subscript ∞ refers to the free-stream conditions, L is a characteristic length for a given flow field, and R_e is the flow Reynolds number defined by;

$$R_e = \frac{\rho_\infty \|u_\infty\| L}{\mu_\infty}. \tag{2.10}$$

The corresponding non-dimensional free-stream values expressed in terms of Mach number and ratio of the specific heats are;

$$\rho_\infty^* = 1, \qquad \|\mathbf{u}_\infty^*\| = 1, \qquad p_\infty^* = \frac{1}{\gamma M_\infty^2}, \qquad i_\infty^* = \frac{1}{(\gamma - 1)\gamma M_\infty^2},$$

and the Mach number is defined according to

$$M = \frac{\|\mathbf{u}\|}{c}. \tag{2.11}$$

Since our fluid equations are cast in non-dimensional form, the same practice is adopted for our turbulence model. Thus, the working variable for our finite volume implementation is χ, the ratio of eddy viscosity to molecular viscosity. In order to obtain realistic inlet boundary conditions for the turbulence variables it is sometimes convenient

to estimate the viscosity ratio. The primary advantage of using the viscosity ratio is that this gives a clear indication of how strong the influence of the turbulent viscosity is compared to the molecular viscosity. Re-casting Equation 2.5 in non-dimensional form using the working variable χ yields;

$$\frac{D\chi}{Dt^*} = c_{b1}S^*\chi - \frac{1}{R_e}\left[c_{w1}f_w\left[\frac{\chi}{\tilde{d}^*}\right]^2\right] + \frac{1}{R_e}\left[\frac{1}{\sigma}\left[\nabla^*\cdot\left((1+\chi)\nabla^*\chi\right) + c_{b2}\left(\nabla^*\chi\right)^2\right]\right]. \qquad (2.12)$$

CHAPTER 3

FINITE VOLUME FORMULATION AND IMPLEMENTATION

3.1 Finite Volume Discretization

After the computational domain is discretized into unstructured cells (hybrid cells are allowed), the finite volume formulation can be written for each control volume. Since our solver is cell-centered, i.e., the control volume is the cell itself, the unknowns are stored at the cell centroids. Following the standard finite volume discretization, we can obtain the semi-discrete form for cell i;

$$\frac{d\mathbf{U}}{dt} + \mathbf{R}(\mathbf{U}) = 0, \tag{3.1}$$

where

$$\mathbf{R}(\mathbf{U}) = \frac{1}{|\Omega_i|} \int_{\partial\Omega_i} \mathbf{H} \cdot \mathbf{n} d\Gamma. \tag{3.2}$$

Here $\mathbf{H}$ includes both inviscid and viscous fluxes, $\mathbf{n}$ is the outward unit normal vector of the faces surrounding cell i and $|\Omega_i|$ is the volume of cell i.

Assuming the control volume is composed of piecewise linear facets, we can use the one-point integration rule for the contour integral. The contour integral above can be replaced with;

$$\mathbf{R}(\mathbf{U}) = \frac{1}{|\Omega_i|} \sum_{k=1}^{n_f} (\mathbf{H} \cdot \mathbf{n} \Delta s)_k, \tag{3.3}$$

where n_f is the number of faces of the control volume and Δs the area of the kth face of cell i.

Remarks.

- The spatial accuracy of the present solver is second-order. The quantities needed to compute the interface fluxes are reconstructed using the solution and the gradient at the cell center. The reconstruction and slope limiting procedure will be addressed in Chapter 4.
- The inviscid flux across the interface is computed via an appropriate approximate Riemann solver. Currently, two options including the HLLC (Harten, Lax, van Leer) [24, 25] flux function or the modified Steger-Warming [27] flux function are available. The HLLC flux function works well for most cases. However, for supersonic flows around blunted bodies, the modified Steger-Warming flux function is recommended to avoid the notorious so-called carbuncle problem [22, 23] of many Riemann solvers.
- The solution gradient across the cell interface is computed via the directional derivative method for viscous fluxes. The details will be provided in Chapter 4. Sutherland's law is used to compute the laminar viscosity which is a function of temperature.

3.2 Data Structure

It is convenient to adopt the face-based data structure for unstructured finite volume solvers. With the face-based data structure, the flux across each face needs to be computed only once and scattered to its two adjacent cells. Two large arrays, IEF(NEF,NE) and IFE(NFE,NF), are used to represent the connectivity between cells and

faces. The IEF array is used to access data on faces from cells. NEF is the number of faces of the cell and NE is the total number of cells. The IFE array is used to access data on cells from faces. NFE is the number of adjacent cells (always = 2 for conforming meshes) of the face and NF is the total number of faces. Using the data structure described here, the solver could handle arbitrarily unstructured meshes. Again, hybrid meshes containing tetrahedral, pyramid, prismatic and hexahedral cells, Figure 1, are allowed.

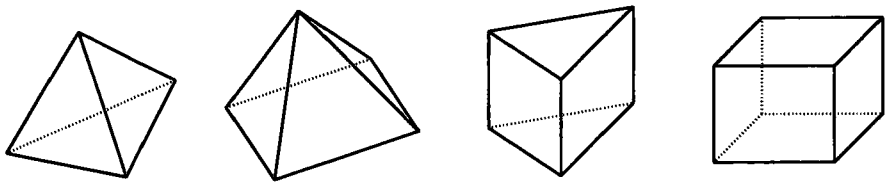

Figure 1: Cell types: tetrahedron, pyramid, prism and hexahedron.

Another data structure IEN(NEN,NE) stores the connectivity information between cells and vertices. NEN is the number of vertices of each cell. The IEN array will be used in interpolating the solution from cell centroids to vertices. Also, the mesh partitioning operates on the IEN array.

3.3 Implicit Time Integration

In our earlier implementation [48, 50], we use the implicit Crank-Nicholson scheme in the time integration. However, our experience shows that the implicit backward Euler formula (BDF) is more stable, at least in our current solver which combines the Jacobian-free GMRES method and the matrix-free LU-SGS preconditioner. Therefore, the BDF method is adopted for the time integration.

$$\frac{\alpha_1 \mathbf{U}^{n+1} + \alpha_0 \mathbf{U}^{n} + \alpha_{-1} \mathbf{U}^{n-1}}{\Delta t} + \mathbf{R}(\mathbf{U}^{n+1}) = 0. \tag{3.4}$$

For first order time accurate scheme (BDF1), $\alpha_1 = 1.0$, $\alpha_0 = -1.0$ and $\alpha_{-1} = 0.0$.

For second order time accurate scheme (BDF2), $\alpha_1 = 1.5$, $\alpha_0 = -2.0$ and $\alpha_{-1} = 0.5$.

We can use **G(U)** to stand for the left hand side of Eq. (3.4). Therefore, Eq. (3.4) can be expressed as;

$$\mathbf{G(U)} = 0. \tag{3.5}$$

To solve Eq. (3.5), which is a non-linear equation system, we use the standard Newton-Raphson iterative method, i.e.;

$$\mathbf{J}\delta\mathbf{U} = -\mathbf{G(U)}, \tag{3.6}$$

where **J** is the Jacobian matrix and can be computed via;

$$\mathbf{J} \equiv \frac{\partial \mathbf{G}}{\partial \mathbf{U}} = \frac{\alpha_{n+1}}{\Delta t}\mathbf{I} + \frac{\partial \mathbf{R}}{\partial \mathbf{U}} = \frac{\alpha_{n+1}}{\Delta t}\mathbf{I} + \tilde{\mathbf{J}}, \tag{3.7}$$

where $\tilde{\mathbf{J}}$ denotes the contribution to **J** from the spatial flux terms and **I** is the identity matrix.

Remarks.

- For steady-state simulations, BDF1 is always used. The user provides a proper Courant-Friedrichs-Levy (CFL) number, the equation for which is given in Chapter 7, and local time stepping technique is applied. A sufficient number of time-steps are given to drive the steady residual (defined as the root-mean-square of **R(U)** in Eq. (3.3) over the whole computational domain) to reach the pre-specified tolerance (measured as order dropped). However, the convergence of aerodynamic force coefficients can often be a better criterion to determine the convergence of complicated flow simulations.

- For unsteady simulations, BDF1 or BDF2 can be chosen. The user provides an appropriate time step. During each physical time step, several sub-iterations are performed to drive the unsteady residual (defined as the root-mean-square of $\mathbf{G}(\mathbf{U})$ in Eq. (3.5) over the whole computational domain) to reach the pre-specified tolerance. A three order drop of the magnitude of the residual during each time step is usually sufficient for the accuracy.

3.4 Jacobian-free GMRES Solver

Inside each non-linear Newton-Raphson iteration, a linear system described as Eq. (3.6) must be solved. This linear system is usually huge, sparse and ill-conditioned. The Generalized Minimal RESidual method (GMRES) [44] has been widely used to solve this kind of linear system. Because the GMRES algorithm involves only matrix-vector multiplication, it is unnecessary to form the Jacobian matrix explicitly. We are able to approximate the matrix-vector product using [45];

$$\tilde{\mathbf{J}}\mathbf{v} \approx [\mathbf{R}(\mathbf{U}+\varepsilon\mathbf{v})-\mathbf{R}(\mathbf{U})]/\varepsilon, \tag{3.8}$$

where $\mathbf{R}$ is evaluated according to Eq. (3.3). Equation (3.8) can be shown to be the first-order Taylor series expansion approximation to the multiplication of the Jacobian matrix, $\tilde{\mathbf{J}}$, and the Krylov vector, $\mathbf{v}$. Obviously, only the spatial contribution $\tilde{\mathbf{J}}$ in Eq. (3.7) needs this approximation because the time-dependent term can be evaluated exactly.

In Eq. (3.8), the choice of ε is a balance between the approximation accuracy and the floating point rounding error. It is as much of an art as a science. We use the following formula to obtain ε;

$$\varepsilon = \delta \frac{\|\mathbf{U}\|_2}{\|\mathbf{v}\|_2}, \tag{3.9}$$

where δ is usually taken as the square root of the machine zero, which is about 10^{-8} to 10^{-7} on most platforms.

This approximation has the following advantages:

- Avoid the difficulty and cost in forming the Jacobian matrix. For high order finite volume solvers, the analytic evaluation of the Jacobian matrix becomes difficult, if not impossible. Furthermore, different Riemann solvers result in different Jacobian matrices. If the code has multiple options of Riemann solvers, the Jacobian matrix has to be coded for each of them.
- Save a significant amount of memory for storing the Jacobian matrix. Although the sparse Jacobian matrix could be stored in a compressed way [44], the storage saving is still significant.

Of course, the disadvantage of not forming the Jacobian matrix is that we have to evaluate the function **R** many times depending on the size of the Krylov space.

3.5 Matrix-free LU-SGS Preconditioning

The convergence of the GMRES algorithm is highly related to the preconditioning. Presently, we adopt the Lower-Upper Symmetric Gauss Seidel (LU-SGS) method [46] as a preconditioning technique. As mentioned in the previous sub-section, it is very difficult and costly to form the Jacobian matrix analytically in high-order finite volume solvers. By contrast, the Jacobian matrix of the low-order (assuming the solution is constant inside the control volume) dissipative flux function can be trivially obtained and is more diagonally dominant and more compact than the high-order Jacobian matrix. Therefore,

the low-order Jacobian matrix is a good candidate as a preconditioner to the high-order Jacobian matrix. We use the simplest and most dissipative flux function, the local Lax-Friedrich (LF) flux [30], to establish the preconditioning matrix. The local LF flux normal to the cell interface can be expressed as;

$$Flux_{LF} = \mathbf{T}^{-1}\left\{\frac{1}{2}[\mathbf{F}(\mathbf{TU}_i)+\mathbf{F}(\mathbf{TU}_j)-\lambda^*(\mathbf{TU}_j-\mathbf{TU}_i)]\right\}, \tag{3.10}$$

where i and j are the indices of the left and right adjacent cells of the face, respectively, $\mathbf{T}$ is the orthonormal rotation matrix of the face, and $\lambda^* = |q^*| + a^*$. Here q^* is the velocity normal to the interface and a^* the speed of sound at the interface. q^* and a^* are determined from the arithmetic average of the left and the right states $(\mathbf{TU}_i + \mathbf{TU}_j)/2$. Thus λ^* represents the largest wave speed in the direction normal to the interface.

For the flux function described as Eq. (3.10), the Jacobian matrix can be easily computed. With the face-based data structure, the resulting Jacobian matrix can be conveniently separated into block diagonal part, lower block triangular part and upper block triangular part, according to Refs. [46, 47], i.e.;

$$\mathbf{J}_{low} = \mathbf{D}+\mathbf{L}+\mathbf{U}, \tag{3.11}$$

where the subscript low indicates that the Jacobian matrix comes from the low-order dissipative flux function. Assuming that $j < i$ in Eq. (3.10), we obtain the $\mathbf{L}$ operator for cell i contributed by cell j;

$$\mathbf{L}_{ij} = \frac{1}{2|\Omega_i|}\mathbf{T}^{-1}\left[\frac{\partial \mathbf{F}(\mathbf{TU}_j)}{\partial(\mathbf{TU}_j)} - \lambda^*\mathbf{I}\right]\Delta s\mathbf{T}, \tag{3.12}$$

where Δs is the area of the face shared by cells i and j. If $j > i$ in Eq. (3.10), then the $\mathbf{U}$ operator is obtained. The diagonal block for row i of $\mathbf{J}_{low}$ can be expressed as;

$$\mathbf{D}_i = \left(\frac{\alpha_{n+1}}{\Delta t} + \frac{1}{2|\Omega_i|} \sum_{k=1}^{n_f} \lambda_l^* \Delta s_l \right) \mathbf{I}, \tag{3.13}$$

which can be represented by a single scalar for each cell. To derive the above equation, we have used the fact that $\sum_{l=1}^{n_f} \mathbf{n}_l \Delta s_l = \mathbf{0}$ for closed polygons. Also, the time dependent term is included in $\mathbf{D}$.

The preconditioning matrix is taken as the approximate Lower Upper-Symmetric Gauss-Seidel (LU-SGS) factorization of $\mathbf{J}_{low}$, namely;

$$\mathbf{P} = (\mathbf{D} + \mathbf{L})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}). \tag{3.14}$$

By comparing Eq. (3.14) and Eq. (3.11), we know $\mathbf{L}\mathbf{D}^{-1}\mathbf{U}$ is dropped in Eq. (3.14). For diagonally dominant and narrowband matrices, $\mathbf{L}\mathbf{D}^{-1}\mathbf{U}$ is negligible. Right-preconditioning is usually adopted in the GMRES algorithm because with right-preconditioning the same residual as the original non-preconditioned system is minimized in the GMRES algorithm. By applying the right-preconditioning to the Jacobian-free GMRES solver, we obtain the new form of Eq. (3.8);

$$\tilde{\mathbf{J}}\mathbf{P}^{-1}\mathbf{v} \approx [\mathbf{R}(\mathbf{U} + \varepsilon\mathbf{P}^{-1}\mathbf{v}) - \mathbf{R}(\mathbf{U})]/\varepsilon. \tag{3.15}$$

It has to be stressed that the function $\mathbf{R}$ in Eqs. (3.3), (3.8) and (3.15) is evaluated following the second order reconstruction procedure. Numerical experience has shown that using the Jacobian matrix resulting from the lower-order, more dissipative flux

function to precondition the Jacobian matrix resulting from higher-order, more accurate flux functions will not compromise the final solution. However, the quadratic convergence rate of the Newton iteration method will be lost. Fortunately, even with this simple preconditioning matrix, the convergence of the GMRES solver can be greatly improved.

Before Eq. (3.15) can be implemented, $\mathbf{v}^{\#} = \mathbf{P}^{-1}\mathbf{v}$ must be solved first. This can be done by solving;

$$\mathbf{P}\mathbf{v}^{\#} = \mathbf{v}, \tag{3.16}$$

for $\mathbf{v}^{\#}$. Substituting Eq. (3.14) into Eq. (3.16) yields;

$$(\mathbf{D}+\mathbf{L})\mathbf{D}^{-1}(\mathbf{D}+\mathbf{U})\mathbf{v}^{\#} = \mathbf{v}, \tag{3.17}$$

which can be solved in two steps in which the block forward sweep;

$$(\mathbf{D}+\mathbf{L})\mathbf{v}^{*} = \mathbf{v}, \tag{3.18}$$

is followed by the block backward sweep;

$$(\mathbf{D}+\mathbf{U})\mathbf{v}^{\#} = \mathbf{D}\mathbf{v}^{*}. \tag{3.19}$$

In Eqs. (3.18) and (3.19), the **L** and **U** operators are computed when needed, thus completely eliminating the need to store the preconditioning matrix. In the parallel version, the LU-SGS preconditioning is implemented locally on each processor to avoid inter-processor communications.

CHAPTER 4

LINEAR RECONSTRUCTION, SLOPE LIMITING, AND FACE GRADIENTS

4.1 Linear Reconstruction

For high-order finite volume schemes, the left and the right states used in the evaluation of interface fluxes must be reconstructed from the solution at cell centers. For second-order finite volume schemes, the reconstruction is linear. Usually, the primitive variables or the characteristic variables, instead of the conservative variables, are reconstructed for best performance. Presently, the reconstruction is based on the primitive variables $\mathbf{q} = (\rho, u_i, e)$ where ρ, u_i and e are density, velocity components, and specific internal energy, respectively. The specific internal energy, e, can be obtained from $e = E - \frac{1}{2} u_i u_i$ where E is the specific total energy. Let us denote the components of the primitive variable vector $\mathbf{q}$ by q. The linear reconstruction can be expressed for cell *0* as;

$$q_k^R = q_0 + \phi \Delta \mathbf{r}_k^T \nabla q_0, \tag{4.1}$$

where q_k^R is the reconstructed solution at the center of the kth face of cell *0*, q_0 is the solution at the cell center, $\Delta \mathbf{r}_k$ is the distance vector between the face center and the cell center, ∇q_0 is the unlimited gradient vector of the primitive variable in cell *0* and ϕ is the slope limiting factor.

To compute the unlimited gradients inside each cell, either the least square method or Gauss theorem can be used. In our previous implementation [48, 50], we used the least square method based on the solution at vertices of the cell. However, we found that the Gauss theorem implementation is more efficient and yields similar results as the least square method. The Gauss theorem states that;

$$\nabla q_0 = \frac{1}{|\Omega_0|}\sum_{k=1}^{n_f} q_k A_k \mathbf{n}_k, \tag{4.2}$$

where q_k is the interpolated solution at the kth face center. q_k is obtained by taking the simple average of the interpolated solutions at the cell vertices. The solution at each vertex is interpolated through pseudo-Laplacian weighted average [50] of solutions at cell centroids surrounding that vertex. A_k and $\mathbf{n}_k$ are the area and outward unit normal of kth face, respectively.

4.2 Slope Limiting

The slope limiter is employed to suppress unphysical overshoots/undershoots and improve the robustness of the solver. An ideal slope limiter should maintain accuracy and should not hamper the convergence rate. The limiting process must also be conservative. The average solution for each cell is kept unmodified after the limiting process. For second-order, cell-centered finite volume solvers, this means that the solution at cell centers should not be changed due to the slope limiting process. We have employed an effective slope limiter for triangular and tetrahedral meshes in [48, 50, 51]. We found that simple extension of that limiter to other types of meshes will introduce excessive dissipation. Therefore, we follow the Local Extremum Diminishing (LED) principle to

design a new limiter that is simple, effective, and suitable for any type of cell. The limiting procedure ensures that no new extrema are allowed during reconstruction.

For compressible flows, density, ρ, is used to compute the slope limiter. This is because density is a good solution smoothness indicator in compressible flow fields. All components of the primitive vector **q** use this same limiter. The computation of the new limiter contains the following steps:

Step 1: Obtain the maximum and minimum density around each vertex.

The maximum and minimum densities around each vertex are obtained by looking at densities at the centroids of all cells surrounding the vertex. The maximum and minimum densities are denoted by $\rho^v_{\max}$ and $\rho^v_{\min}$, respectively. The superscript 'v' stands for vertex.

Step 2: Obtain the maximum and minimum solution around each cell.

The maximum and minimum densities around each cell are obtained by;

$$\rho^c_{\max} = \max_i(\rho^v_{i,\max}), \text{ and} \tag{4.3a}$$

$$\rho^c_{\min} = \min_i(\rho^v_{i,\min}), \tag{4.3b}$$

where the superscript 'c' stands for cell and 'i' is the index of vertices of the cell.

Step 3: Compute the allowable variation of the density.

The allowable density variation of each cell is computed via;

$$\Delta\rho^c_{\max} = \max(\rho^c_{\max} - \rho_o, \varepsilon\rho_o), \text{ and} \tag{4.4a}$$

$$\Delta\rho^c_{\min} = \min(\rho^c_{\min} - \rho_o, -\varepsilon\rho_o). \tag{4.4b}$$

The beauty of the above formulation is that they allow some small amount of numerical noise by adjusting the positive parameter ε. This is important because we do not want the limiter to be active in regions where the solution is smooth with negligible numerical noise. Numerical experience shows that $\varepsilon = 10^{-3} - 10^{-4}$ is sufficient to suppress unphysical overshoots/undershoots while not affecting the residual convergence. Also, note that the computed $\Delta\rho^c_{\max}$ and $\Delta\rho^c_{\min}$ are always positive and negative, respectively.

Step 4: Compute the slope limiting factor.

Using the unlimited density gradient computed according to Eq. (4.2), we can calculate the unlimited variation of density at each vertex of cell *0*.

$$\Delta\rho^v_i = \Delta\mathbf{r}_i^T \nabla\rho_0. \tag{4.5}$$

We compare each $\Delta\rho^v_i$ with $\Delta\rho^c_{\max}$ and $\Delta\rho^c_{\min}$. The limiting factor limits $\Delta\rho^v_i$ to be within the range defined by $\Delta\rho^c_{\max}$ and $\Delta\rho^c_{\min}$. It can be expressed as;

$$\phi_i = \begin{cases} \dfrac{\Delta\rho^c_{\max}}{\Delta\rho^v_i} & \text{if} \quad \Delta\rho^v_i > \Delta\rho^c_{\max} \\ \dfrac{\Delta\rho^c_{\min}}{\Delta\rho^v_i} & \text{if} \quad \Delta\rho^v_i < \Delta\rho^c_{\min} \\ 1 & \text{otherwise} \end{cases}. \tag{4.6}$$

As can be seen, $0 < \phi_i \le 1$. The final limiter for the cell is obtained by taking the minimum value of ϕ_i, i.e., $\phi = \min_i(\phi_i)$, because we want the solution at all vertices of the cell to satisfy Eq. (4.6). ϕ will be applied in Eq. (4.1) to reconstruct the primitive solution at face centers.

4.3 Face Gradients

The discretization of viscous terms requires the gradient information of primitive variables across the cell interfaces. For unstructured meshes, there is no explicit line structure as in structured meshes. Therefore, obtaining the gradient of a primitive variable, q, requires special attention. Figure 2 provides a stencil of the face gradient.

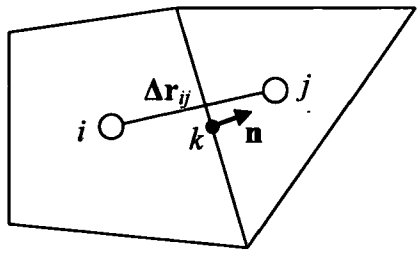

Figure 2: Stencil to compute the gradient at the face.

Mathur and Murthy [52] adopted a sophisticated formula to compute the gradient at the face k.

$$\nabla q_k = \frac{q_j - q_i}{\Delta \mathbf{r}_{ij} \cdot \mathbf{n}} \mathbf{n} + (\overline{\nabla q}_k - \frac{\overline{\nabla q}_k \cdot \Delta \mathbf{r}_{ij}}{\Delta \mathbf{r}_{ij} \cdot \mathbf{n}} \mathbf{n}), \tag{4.7}$$

where $\overline{\nabla q}_m = \frac{1}{2}(\nabla q_i + \nabla q_j)$, $\Delta \mathbf{r}_{ij}$ is the displacement vector connecting cell centroids i and j, and $\mathbf{n}$ is the unit normal of the face k. The idea behind Eq. (4.7) is that the gradient at the face is divided into two components. The first component is the gradient in the direction normal to the face and computed by the first term of Eq. (4.7). The second component is computed by averaging the gradients at cell centroids and removing the component normal to the face.

A slightly different formula is used by Hyams et al. [53];

$$\nabla q_k = \frac{q_j - q_i}{\| \Delta \mathbf{r}_{ij} \|} \mathbf{n}_{ij} + (\overline{\nabla q}_k - \frac{\overline{\nabla q}_k \cdot \Delta \mathbf{r}_{ij}}{\| \Delta \mathbf{r}_{ij} \|} \mathbf{n}_{ij}), \tag{4.8}$$

where $\mathbf{n}_{ij} = \Delta \mathbf{r}_{ij} / \| \Delta \mathbf{r}_{ij} \|$, the direction of the displacement vector connecting cell centroids *i* and *j*, which is different than **n** in Eq. (4.7), which is the direction normal to the face. Obviously, if $\mathbf{n}_{ij}$ is in the same direction as **n**, Eq. (4.7) and Eq. (4.8) will be identical. Our numerical experience shows that these two formulations yield very similar aerodynamic forces for a typical unstructured mesh.

CHAPTER 5

SPALART-ALLMARAS DETACHED EDDY SIMULATION (SA-DES) IMPLEMENTATION

In Chapter 2, we presented the SA-DES [38-42] turbulence model used in our current solver. In this chapter, we describe our SA-DES turbulence model implementation. The solution scheme (i.e. discretization, data structure, implicit time integration, etc.) used to solve the SA-DES turbulence model, Eq. (2.12), is identical to the scheme adopted for the Navier-Stokes equations, which is detailed in Chapters 3 and 4. Inside each physical time step, the Navier-Stokes equations are solved first. Then the turbulence equation is solved. This loose coupling strategy helps the convergence of both fluid and turbulence. In addition, to ensure stability and robustness, a threshold can be applied to both the SA-DES model functions, and the computed turbulent eddy viscosity, which is discussed in the following sub-sections.

5.1 SA-DES Function Limiting

Figure 3 depicts Eq. (2.8a).

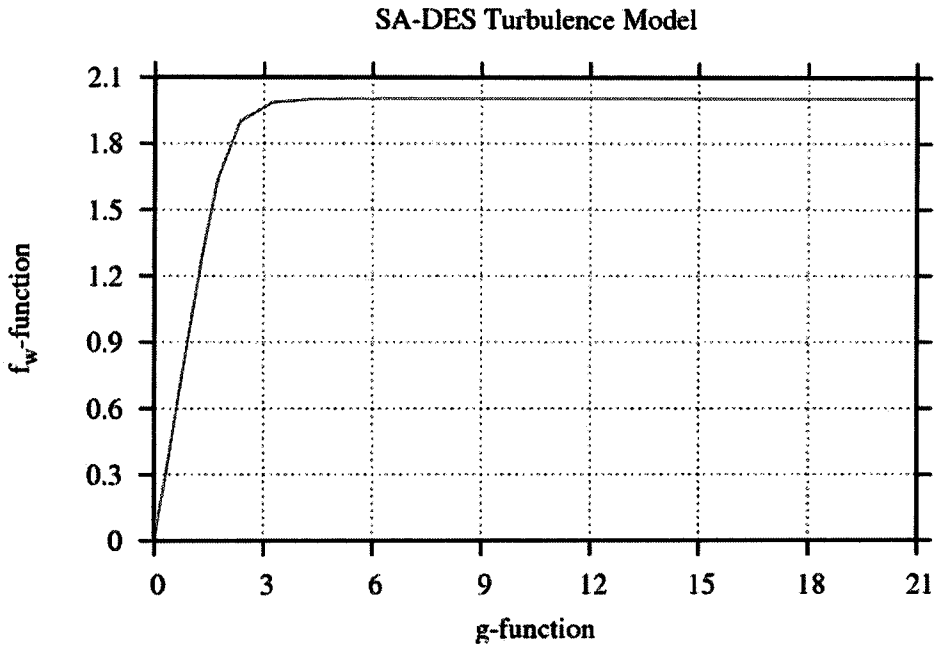

Figure 3: Spalart-Allmaras Detached Eddy Simulation Implementation: f_w-function versus g-function.

As can be seen from the graph, the function f_w is asymptotic, varying only at values of approximately $0.0 \leq g \leq 4.0$. This range of the function g corresponds to $0 \leq r \leq 1.5$, which can be seen in Figure 4.

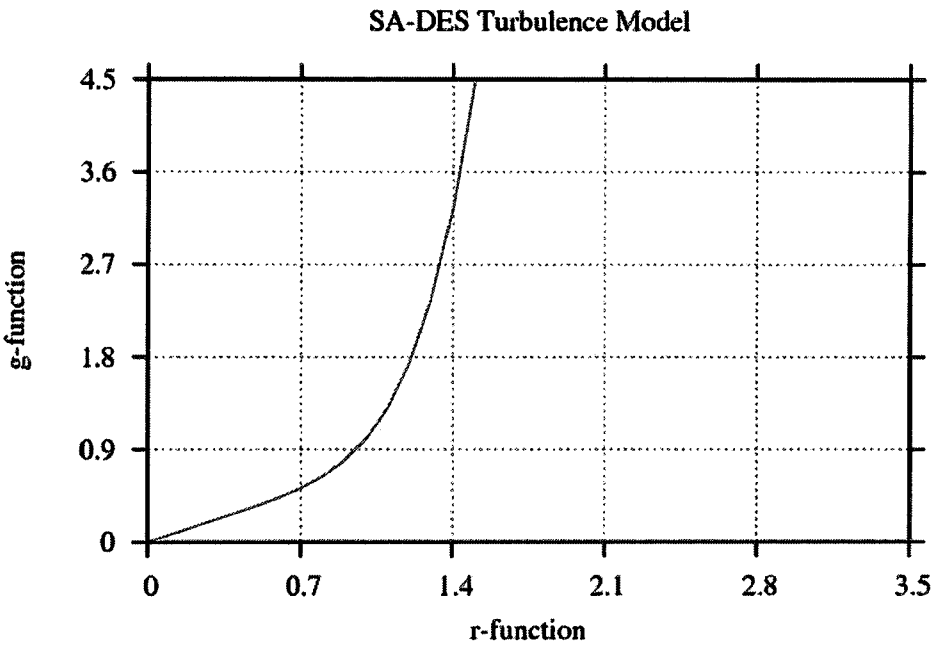

Figure 4: Spalart-Allmaras Detached Eddy Simulation Implementation: *g*-function versus *r*-function.

Using this information, numbers exceeding the machine accuracy, which can happen for large r due to the r^6 term of Eq. (2.8b), can be avoided when calculating the function g. Hence if $r \geq 1.5$, then we enforce $g = 4.0$, and consequently $f_w \approx 2.0$. Conversely, the possibility for numbers that approach and in some instances exceed machine accuracy is present for extremely small r as well. To avoid this occurrence, we use linear curve fitting to approximate the value of f_w as $0.7^* r$ if $r \leq 0.01$. The reasoning for this approximation is evident from Figures 5 and 6.

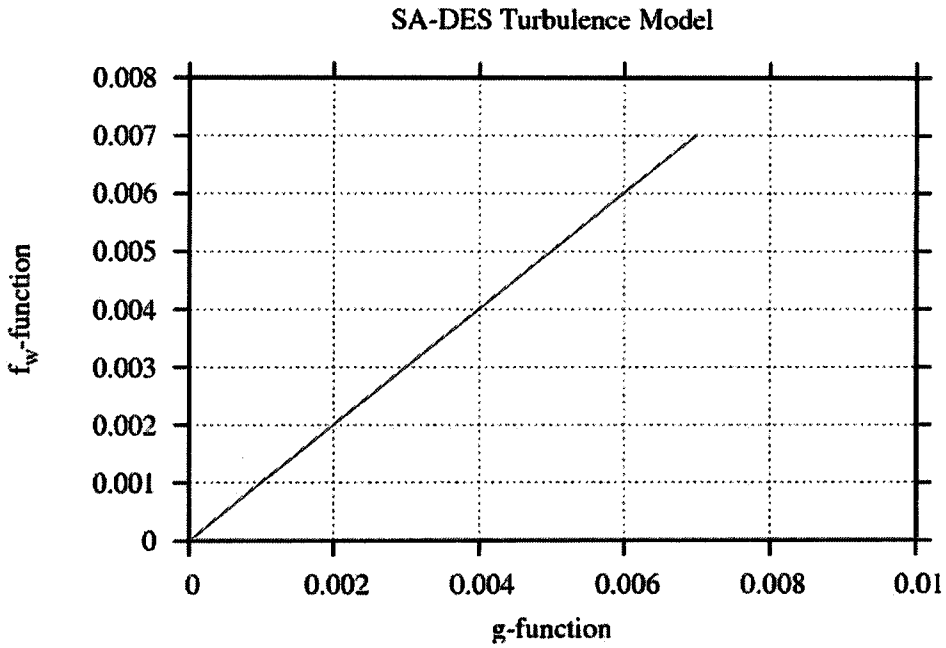

Figure 5: Spalart-Allmaras Detached Eddy Simulation Implementation: f_w-function versus g-function.

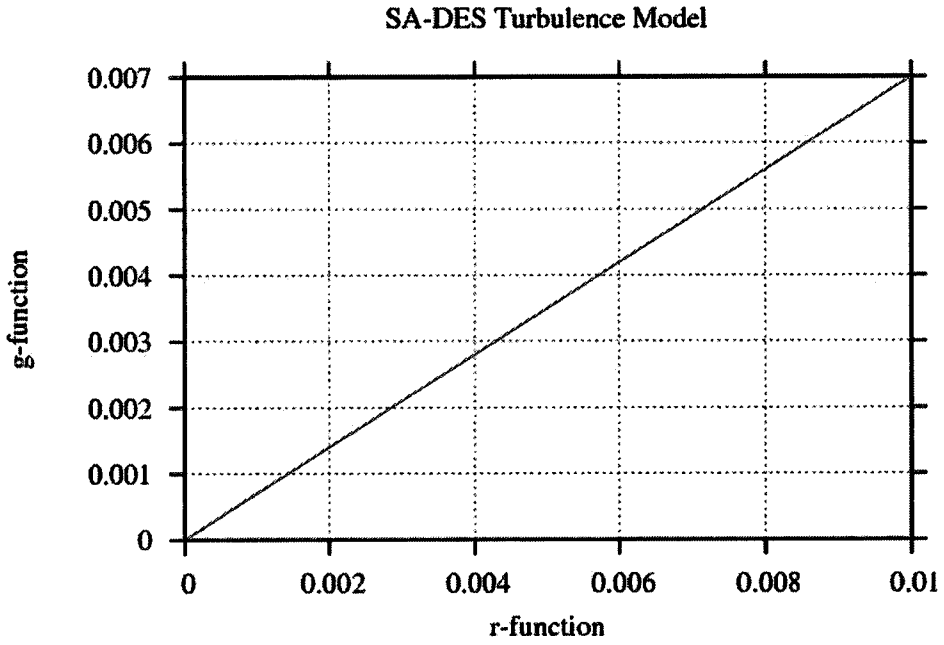

Figure 6: Spalart-Allmaras Detached Eddy Simulation Implementation: g-function versus r-function.

5.2 SA-DES Viscosity Ratio Limiting

Often, negative, non-physical numbers can arise within computations. We wish to avoid operating on these negative numbers, especially in the Riemann solver, and maintain positivity. However when this instance is discovered, it is too late to modify the solution gradient at the current time-step. This non-physical state may be the result of applying a limiter that is based only on density, ρ, to the reconstructed primitive variables. Thus, a threshold is set on the reconstructed viscosity ratio. In addition, we wish to prevent excessively large values of the viscosity ratio. As such, the currently imposed viscosity ratio threshold is $10.e^{-15} \leq \chi \leq 2000$. If a particular solution of χ exceeds this range, χ is returned to the solution at the preceding time step. Numerical experience has shown this range to be acceptable for low subsonic to moderate supersonic flow regimes.

CHAPTER 6

PARALLELIZATION AND VECTORIZATION

This chapter discusses implementation details about the parallelization and vectorization for efficiency in large scale simulations.

6.1 Parallelization

Our solver was first parallelized on the Cray T3E-1200 using the ParMETIS mesh partitioning [54] and the Message Passing Interface (MPI) parallel programming module. After the mesh partitioning, each processor will be assigned a work load of similar size. There is no so-called *master processor*. However, there is a single node assigned to be responsible for the file I/O and error message output. Due to the data structures we use, we have communication requirements for vertices, faces and cells. The gather/scatter communications subroutines for faces and cells are written directly based on those for nodes, which have been extensively used in our finite element solvers [8-19]. The inter-processor subroutines are extensions from our finite-element-based solvers adapted to the face-based data structure in the current finite volume solver. Due to the mesh partitioning, only nodes, faces and cells on the partition boundaries involve the inter-processor communication. Very efficient non-blocking (MPI) functions are called to set up the inter-processor "gather" and "scatter" routines in the pre-processing stage. As will be seen in Chapter 7, the parallel scalability is excellent.

6.2 Vectorization

The current solver has also been fully vectorized on the Cray X1E parallel/vector/multi-streaming architecture. The integrated parallel/multi-streaming/vector system of the Cray X1E provides a high level of computational performance that can be achieved only if all computations are fully vectorized and multi-streamed. In the real CFD computation, the mesh is first partitioned and each partition is assigned to a single MSP. The "gather" and "scatter" communication subroutines will be responsible for the communication and coordination among the MSPs through the X1E's interconnect network. Inside each MSP, the compiler will try to multi-stream and vectorize each loop. For the loop to be successfully vectorized by the compiler, the loop must be free of any of the following: data dependencies, memory contention, I/O statements, non-inline calls to subroutines and functions. The very useful compiler option "`-rm`" can be used to prompt the compiler to generate an `.lst` file for each source file. The `.lst` file reports the multi-steaming and vectorization status of each loop in the source file. If the loop is fully multi-streamed and vectorized, each line of the loop will be marked with "MV" at the beginning of that line.

The face-based data structure is used to compute the inviscid and viscous fluxes across each face. The resultant flux vector is then scattered to the two adjacent cells of the face. The "add" operation assembles the global cell-based vector composed of the residual. Therefore, the Cray X1E compiler will not vectorize loops such as this by default because of the memory scatter statements, and if we force the compiler to vectorize the loop, it is possible that two faces access the same cell simultaneously, causing memory contention.

Another situation that prohibits vectorization is the matrix-free LU-SGS preconditioner. The LU-SGS preconditioner involves solving two block triangular linear equation systems. However, exact solution of triangular systems requires sequential computations.

Note that the face loop and the LU-SGS preconditioner are extensively called by the GMRES solver. These two situations consume the vast majority of the total computational time. These two situations are coded into two subroutines respectively. Therefore, the vectorization of these two subroutines is crucial to achieve optimal performance of the code on the Cray X1E. To eliminate the memory contention caused by the "assemble" operation at the end of the face loop, our strategy is to use the so-called *face-coloring algorithm*. To avoid the sequential computations in solving the triangular system, we apply the approximate truncated Neumann expansions of the inverse of the triangular matrices.

6.2.1 Face Grouping and Cell Grouping

To vectorize the face loops, we divide the faces into groups. Inside each group, no two faces have the same adjacent cell. Thus, the memory contention problem can be avoided. The face grouping, also known as a face-coloring scheme, is similar to the element-coloring scheme used in vectorizing the node-based finite element solvers. This grouping process is done in the pre-processing stage. The grouping algorithm is quite simple with the current data structure. With this algorithm, face groups are created to contain as many faces as possible. Once the faces are grouped, we will modify the face loop to contain an outer group loop and an inner face loop. Since we have guaranteed that there will be no memory contention inside each face group, we can force the vectorization of the inner

face loop by applying the "CONCURRENT" compiler directive. This directive is used in place of the more standard "IVDEP" directive found on older Cray systems.

The pseudo code of the new face loop is shown below:

```
DO IG = 1, NGF  ! NGF is the number of face groups.
   IFACE_BEG = FGROUP(IG)
   IFACE_END = FGROUP(IG+1) - 1

   !DIR$ CONCURRENT
   DO IFACE = IFACE_BEG, IFACE_END
      IE1 = IFE(1,IFACE)
      IE2 = IFE(2,IFACE)
      ...
      ... ! perform computations, e.g., fluxes.
      ...
      ... ! scattered to two adjacent cells.
   ENDDO
ENDDO
```

The cells are also grouped according to the *element-coloring algorithm*. Inside each cell group, no two cells will access the same vertex. This expedites two processes; pseudo-Laplacian interpolation from cell centroids to vertices and finding the maximum and minimum solutions at vertices. Both processes are done by looping over cells and using array IEN which stores the connectivity information between cells and vertices. The cell grouping does not play as important a role as the face grouping, because these two processes are called only once within each Newton-Raphson nonlinear iteration. By contrast, the face loop is called extensively inside the GMRES solver because we use a Jacobian-free method.

Due to the cell and face grouping, the arrays IEN, IEF and IFE described in Chapter 3 must be correspondingly modified to account for the index changes. For pure tetrahedral meshes, the face coloring algorithm will divide all faces into 7 groups on average. The element coloring algorithm will divide all cells into about 42-45 groups. Figure 7 shows

the grouping statistics in a typical processor for a tetrahedral mesh containing 3,652,436 cells and 7,347,956 faces and partitioned by 24 processors.

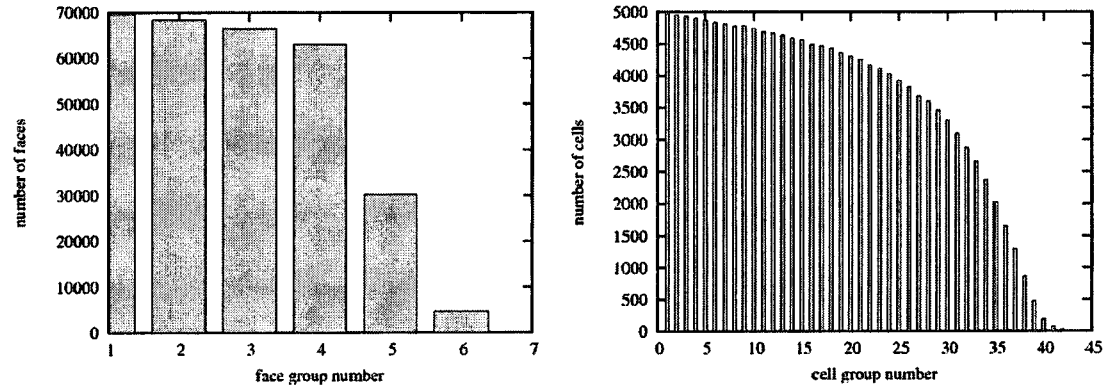

Figure 7: Face and cell grouping statistics about a pure tetrahedral mesh. ne = 3,652,436, nproc = 24.

Figure 8 shows the face and cell grouping statistics information in a typical processor about a pure hexahedral mesh.

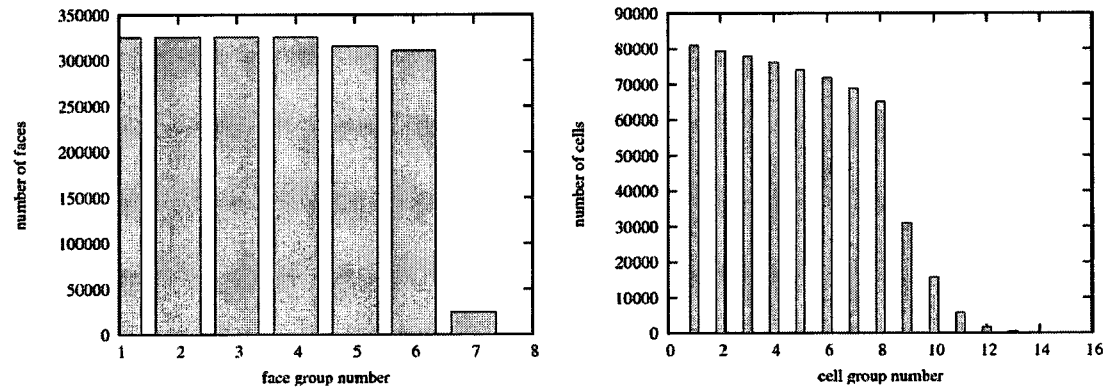

Figure 8: Face and cell grouping statistics about a pure hexahedral mesh. ne = 40,151,112, nproc = 64.

This mesh contains 40,151,112 hexahedra and 120,604,576 faces. 64 processors are used to partition this mesh. On each processor, the cells are divided into 14-17 groups and the faces are divided into 8-9 groups.

As can be seen from Figures 7 and 8, on average, the majority of the groups contain a large number of faces or cells, with a few smaller groups for the remainder of faces or

cells. The number of faces or cells in each group determines the "vector length" for the face or cell loops. These sizes are, of course, much larger than the preferred minimum vector-length size. With the full vectorization of the face or cell loops, high-sustainable performance can be achieved. Information on the porting, optimization, and performance of a finite element code on the Cray X1E has been reported by Johnson [19].

6.2.2 Vectorization of the LU-SGS Preconditioner

To vectorize the LU-SGS preconditioner, we apply the truncated Neumann expansions of the inverse of triangular matrices. For example, the Neumann expansion of the inverse of the lower triangular matrix is given by;

$$(\mathbf{I}+\mathbf{L})^{-1} = \sum_{k=0}^{n-1} (-1)^k \mathbf{L}^k. \tag{6.1}$$

With the truncated Neumann expansions, the sequential computations needed to solve the triangular system exactly can be reduced to a finite number of matrix-vector multiplications, i.e., the $\mathbf{L}$ operator in Eq. (6.1).

To utilize Eq. (6.1), we must rewrite the preconditioning matrix given by Eq. (3.14) in the following form;

$$\mathbf{P} = (\mathbf{I}+\hat{\mathbf{L}})\mathbf{D}(\mathbf{I}+\hat{\mathbf{U}}), \tag{6.2}$$

where $\hat{\mathbf{L}} = \mathbf{L}\mathbf{D}^{-1}$ and $\hat{\mathbf{U}} = \mathbf{D}^{-1}\mathbf{U}$ .Correspondingly, the forward sweep and the backward sweep become;

$$(\mathbf{I}+\hat{\mathbf{L}})\mathbf{v}^* = \mathbf{v}, \tag{6.3}$$

and

$$(\mathbf{I}+\hat{\mathbf{U}})\mathbf{v}^{\#}=\mathbf{D}^{-1}\mathbf{v}^{*}, \tag{6.4}$$

respectively.

Remember the diagonal matrix **D** is a scalar constant for each cell, thus producing no difficulty in the vectorization.

If the first two terms of the Neumann expansion are kept, then $\mathbf{v}^{*}$ can be computed via;

$$\begin{aligned}\mathbf{v}^{*} &= \mathbf{v}-\hat{\mathbf{L}}\mathbf{v},\\ &= \mathbf{v}-\mathbf{L}(\mathbf{D}^{-1}\mathbf{v}).\end{aligned} \tag{6.5}$$

If we want to keep the first three terms, then we have;

$$\begin{aligned}\mathbf{v}^{*} &= \mathbf{v}-\hat{\mathbf{L}}\mathbf{v}+\hat{\mathbf{L}}^{2}\mathbf{v},\\ &= \mathbf{v}-\mathbf{L}(\mathbf{D}^{-1}(\mathbf{v}-\mathbf{L}(\mathbf{D}^{-1}\mathbf{v}))).\end{aligned} \tag{6.6}$$

Similar expressions can be obtained for $\mathbf{v}^{\#}$. Obviously, the vectorization can be easily achieved using Eqs. (6.5) or (6.6).

Numerical experiments show that keeping the first two terms of the Neumann expansion is sufficient to yield satisfactory convergence results.

CHAPTER 7

NUMERICAL EXAMPLES

In this chapter we present 2-dimensional and 3-dimensional examples to demonstrate the capability and performance of our matrix-free, parallel and vectorized, unstructured finite volume solver. These examples include:

- Subsonic Flow past a NACA0015, Square-Tip Airfoil in 2-Dimensions.
- Subsonic Flow past a NACA0015, Square-Tip Airfoil in 3-Dimensions.
- 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds.
- Scalability Investigations using Unstructured Grids on Vector/Multi-Streaming and Linux Cluster Platforms.

At this point, it should again be stressed that the intended use of the current finite solver is the simulation of transient compressible viscous flows; subsonic, transonic, and supersonic in nature. Applications to flows within the subsonic regime, below the incompressible limit, are beyond the scope of this research. However, the results of example cases one and two, subsonic flow past a NACA0015, do provide some confidence in the solvers capability of treating some subsonic flows below Mach number 0.3. In addition, the absence of any mechanism to treat the dissociation and ionization of air molecules, which is present in hypersonic flows, makes such applications equally undesirable. As such, ideal applications will correspond to a Mach number range of

approximately 0.3 - 3.0. The low-memory, matrix-free iterative technology of this implicit finite volume solver makes the *transient* simulation of flows within the stated range possible, across multiple HPC platforms without the loss of scalability. This in turn, allows for maximum usage of available computational resources. This indeed, is the primary benefit of this finite volume solver as compared to other commercially available CFD software.

All grids used were created using *GRIDGEN* Version 15 software (Pointwise Inc.). To accurately predict the aerodynamic forces, the grid around the aerodynamic body must be designed carefully. One of the primary requirements is generating high aspect ratio cells near the body. The first layer thickness is a function of the Reynolds number, i.e.;

$$y^+ = 0.172 y^* R_e^{0.9}, \tag{7.1}$$

where y^* is the non-dimensional first layer thickness next to the body and R_e is the flow Reynolds number. $y^+ \approx 1$ is assumed in the first layer [55]. Therefore, according to Eq. (7.1), the first layer thickness of the mesh can be computed.

Simulations were performed on the Cray X1E and the Linux Network Evolocity II cluster, JVN. To obtain transient solutions more quickly, all simulations are treated as steady state computations initially, and later as time accurate computations to allow the unsteady features of the flow to develop. Performance statistics are given for each example to demonstrate the efficiency, scalability, and speed of the present solver. The total CPU time excluding the problem setup and preprocessing time for the parallel simulation is recorded in the simulation. The speed, T_c, is evaluated according to the following formula;

$$T_c = \frac{n_{proc}}{n_{elem} n_{ts} n_{it} n_k} T_{run}, \tag{7.2}$$

where n_{proc}, n_{elem}, n_{ts}, n_{it}, and n_k are the numbers of processors, elements, time steps, nonlinear iterations within each timestep, and size of Krylov space, respectively, and T_{run} is the CPU time. Additionally, numerical and experimental aerodynamic coefficients are given to demonstrate the accuracy of the present solver.

7.1 Subsonic Flow past a NACA0015, Square-Tip Airfoil in 2-Dimensions

A NACA0015, square-tip airfoil is subjected to low speed, viscous flow at Mach 0.1235 and 12 degree angle of attack. The airfoil chord-length is 1.0 unit. The inflow boundary is located 10 chords away from the leading edge of the airfoil, whereas the outflow boundary is 15 chords away from the trailing edge. The resulting grid contains 11,100 quadrilateral cells adjacent to the airfoil surface and 48,862 triangular cells. There are 222 points along the airfoil surface. The first layer thickness is approximately 1.e-5, which coincides with a y+ of approximately 1 based on the Reynolds number, 1.5e+6. The maximum aspect ratio, which is defined as the ratio of the average length and average width, is 2993 for the first quadrilateral layer adjacent to the airfoil surface. An image of the grid used for this simulation is shown in Figure 9.

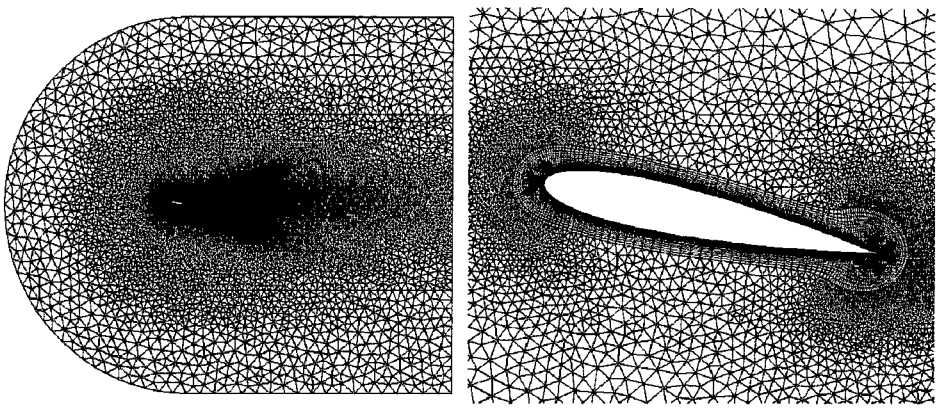

Figure 9: Subsonic Flow past a NACA0015, Square-Tip Airfoil in 2-Dimensions: hybrid grid generated for NACA0015, square-tip airfoil.

The time accurate duration of this computation was 17.75 characteristic seconds. The time-cost of our solver for this application, according to Eq. (7.1), is 3.884113e-6 seconds. The experimental lift and drag coefficients for the NACA0015 airfoil at 12 degree angle of attack are 1.13 and 0.01735, respectively. Our computations yield a lift and drag coefficient of 1.097 and 0.025 respectively. The pressure coefficient, C_P, along the airfoil surface is shown in Figure 10.

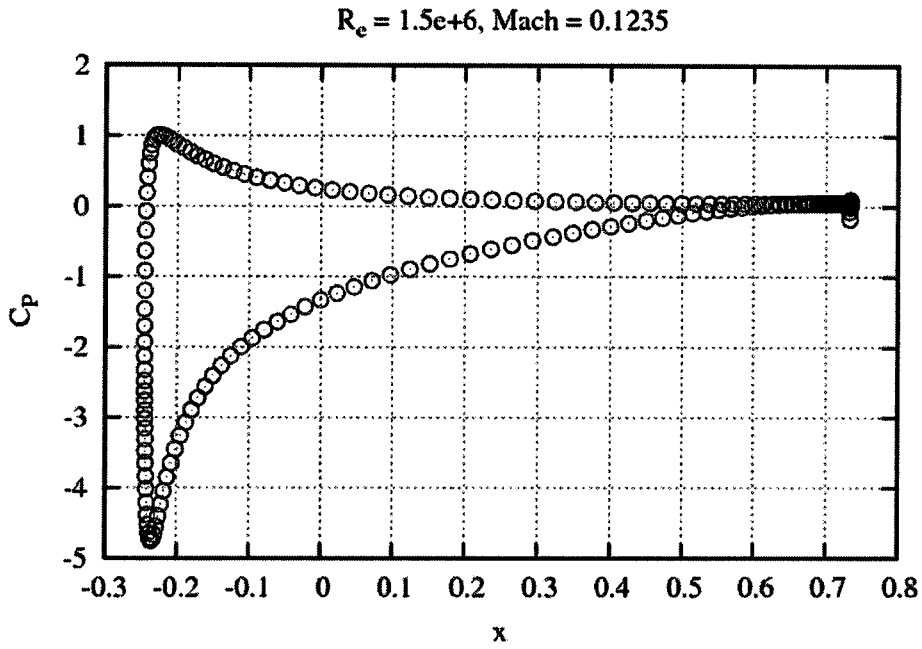

Figure 10: Subsonic Flow past a NACA0015, Square-Tip Airfoil in 2-Dimensions: pressure coefficient along the airfoil surface.

Also, the skin-friction coefficient, C_f, can be seen in Figure 11.

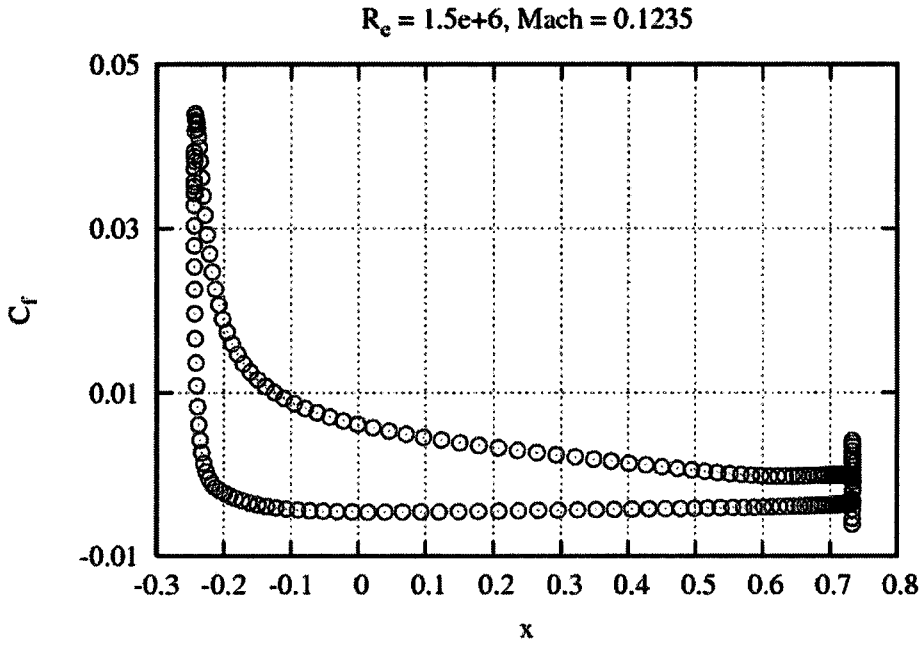

Figure 11: Subsonic Flow past a NACA0015, Square-Tip Airfoil in 2-Dimensions: skin-friction coefficient along the airfoil surface.

Finally, the mach and pressure contours can be seen in Figure 12.

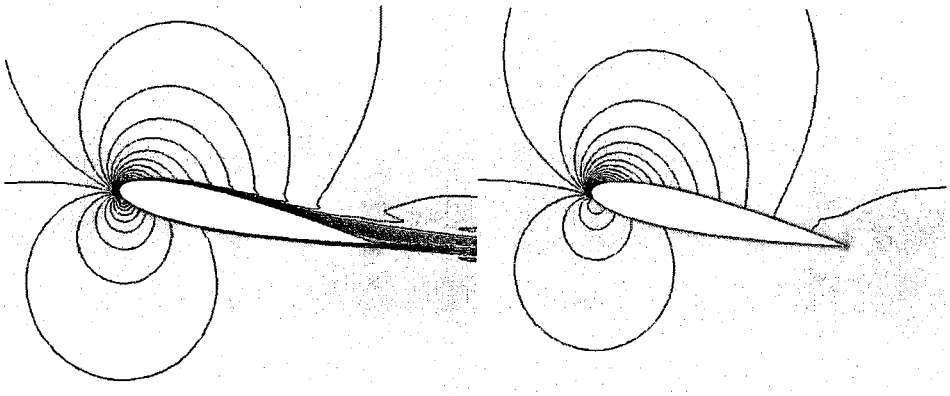

Figure 12: Subsonic Flow past a NACA0015, Square-Tip Airfoil in 2-Dimensions: Mach contours, left, and pressure contours, right.

7.2 Subsonic Flow past a NACA0015, Square-Tip Wing in 3-Dimensions

Here, a new 2-dimensional hybrid grid was generated for the NACA0015, square-tip airfoil, and then extruded in the span-wise direction, thus creating a constant cross-section 3-dimensional grid consisting of prism and hexahedral elements. For the new 2-dimensional grid, the mesh refinement was confined to a smaller region around the airfoil. Again, the airfoil chord-length is 1.0 unit, and the wing span-length is 5.59/1.7 units. The inflow boundary is located 10 chords away from the leading edge of the airfoil, whereas the outflow boundary is 15 chords away from the trailing edge. The far field boundary is located 10 chords away from the wing tip. The resulting grid consisted of 16,130,184 nodes and 26,630,656 elements, in which 21,264,666 elements are prisms and 5,365,990 elements are hexahedral. An image of the grid used for this simulation is shown in Figure 13.

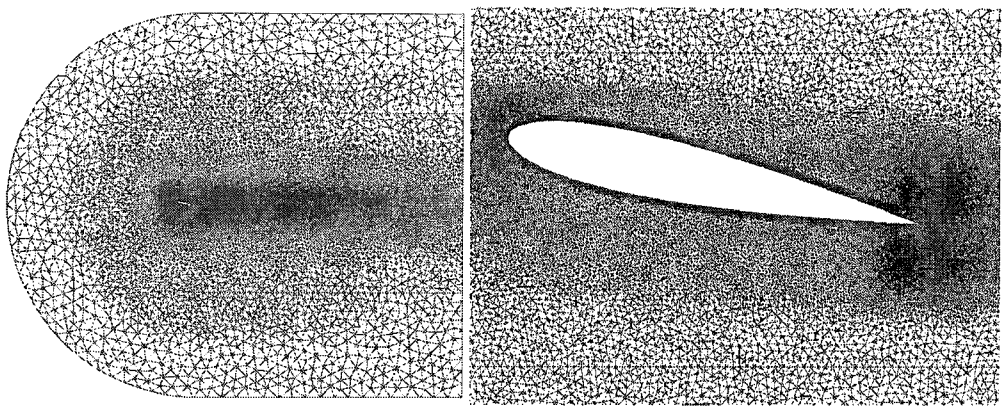

Figure 13: Subsonic Flow past a NACA0015, Square-Tip Wing in 3-Dimensions: cross-section of hybrid grid for NACA0015, square-tip wing.

Computations were done on both the Cray X1E and the JVN. The results are summarized below.

7.2.1 JVN Computations

Steady state computations were done on the Linux cluster located at the Army Research Laboratory. To accelerate the convergence toward steady solutions, local time stepping [11] is used. The time step for cell i is determined according to;

$$\Delta t_i \le \mathrm{CFL} \frac{\Omega_i}{\sum_k \Delta s_k \left| \min\left(\mathbf{u}\cdot\mathbf{n} - a, 0\right)\right|}. \tag{7.2}$$

Here $\mathbf{u}$ and a are evaluated at the centroid of the element for simplicity, k is the index of the faces surrounding the element, Δs_k is the area of the face and Ω_i is the volume of the element. The CFL number is gradually increased for rapid convergence as the flow is progressed toward solution.

One nonlinear iteration is performed for the Navier-Stokes equations and one non-linear iteration for the Spalart-Allmaras turbulence equation per time step. The Krylov space is set to 10 in the GMRES solver. Approximately 5000 time steps were done using

32 – 128 processors, depending on machine node availability. The total cpu time for this problem is approximately 96 hours.

Figure 14 and 15 show the lift coefficient and drag coefficient versus the time step, respectively.

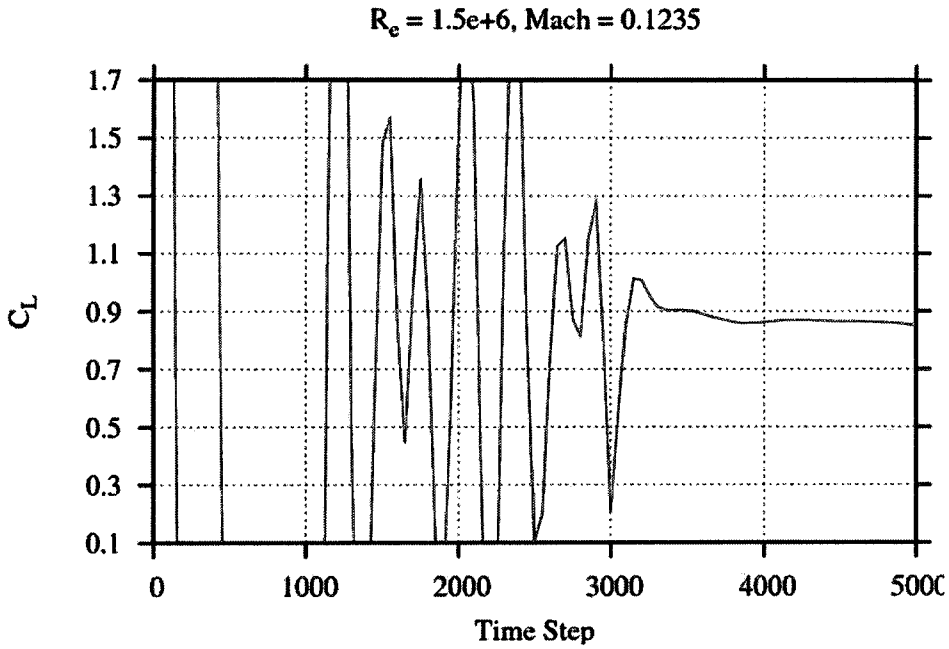

Figure 14: Subsonic Flow past a NACA0015, Square-Tip Wing in 3-Dimensions: steady state lift coefficient versus time step.

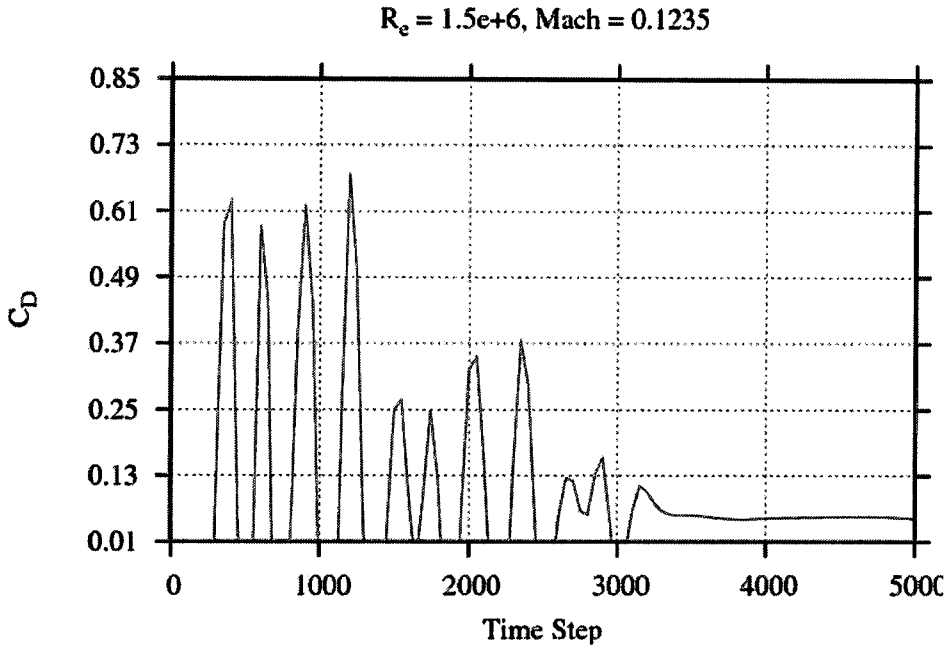

Figure 15: Subsonic Flow past a NACA0015, Square-Tip Wing in 3-Dimensions: steady state drag coefficient versus time step.

7.2.2 Cray X1E Computations

Time accurate computations were done on the Cray X1E. The non-dimensional time step used for these computations is 0.001. Thus, 1000 time steps are required for 1 characteristic second. The calculations were carried out for 15,000 time steps, i.e. 15 characteristic seconds.

Two nonlinear iterations were performed for the Navier-Stokes equations and one nonlinear iteration for the Spalart-Allmaras turbulence equation per time step. The Krylov space is set to 10 in the GMRES solver. 64 MSPs were used, resulting in each time step requiring only 17 seconds. The total cpu time for this problem is 70 hours. It should be noted that using 256 MSPs of the same machine, the computation will take less than 20 hours.

Figure 16 and 17 show the lift coefficient and drag coefficient versus the characteristic time, respectively.

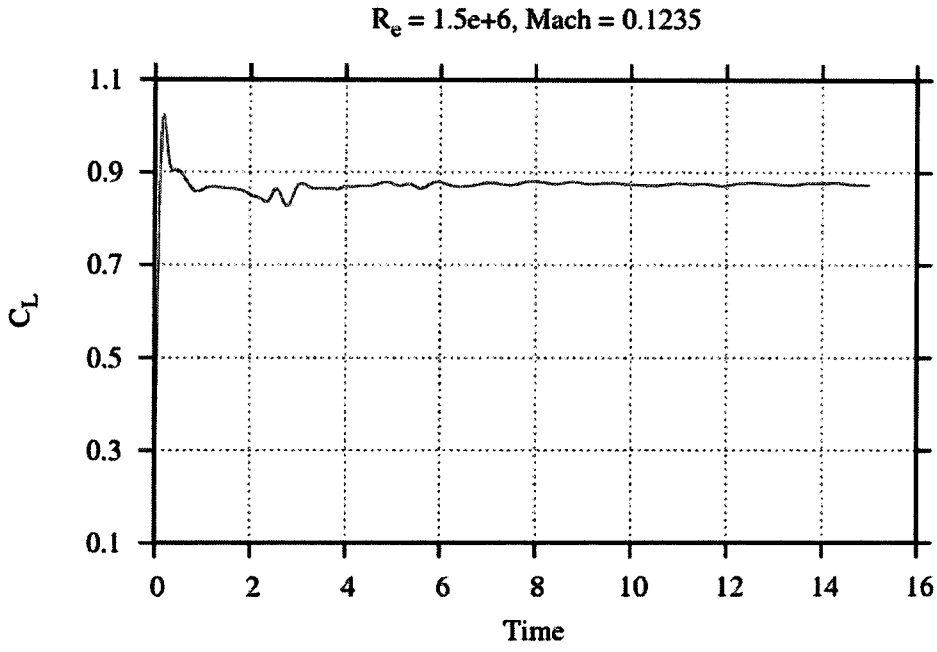

Figure 16: Subsonic Flow past a NACA0015, Square-Tip Wing in 3-Dimensions: unsteady lift coefficient versus characteristic time.

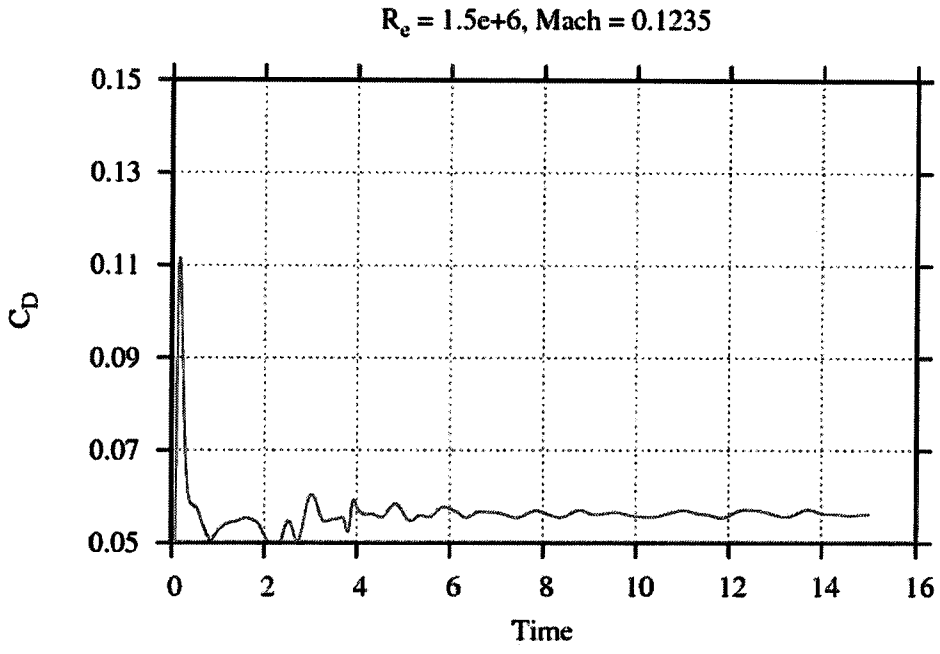

Figure 17: Subsonic Flow past a NACA0015, Square-Tip Wing in 3-Dimensions: unsteady drag coefficient versus characteristic time.

In Figures 18 - 21, the flow field pressure distribution is shown at varying locations in the span-wise direction.

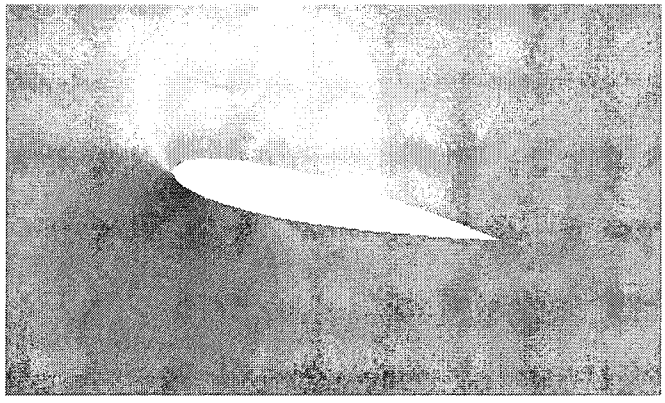

Figure 18: Subsonic Flow past a NACA0015, Square-Tip Wing in 3-Dimensions: pressure distribution at 10% span.

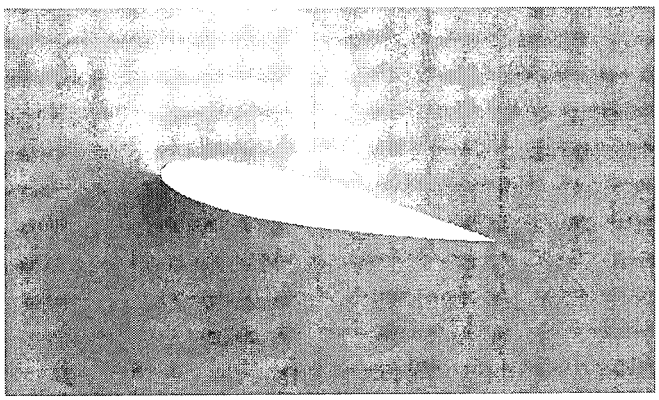

Figure 19: Subsonic Flow past a NACA0015, Square-Tip Wing in 3-Dimensions: pressure distribution at 50% span.

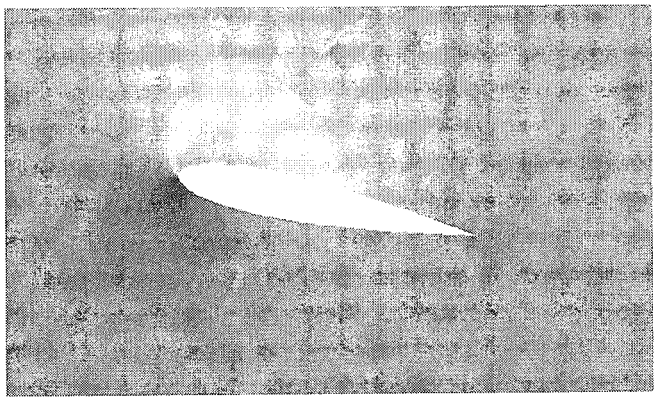

Figure 20: Subsonic Flow past a NACA0015, Square-Tip Wing in 3-Dimensions: pressure distribution at 90% span.

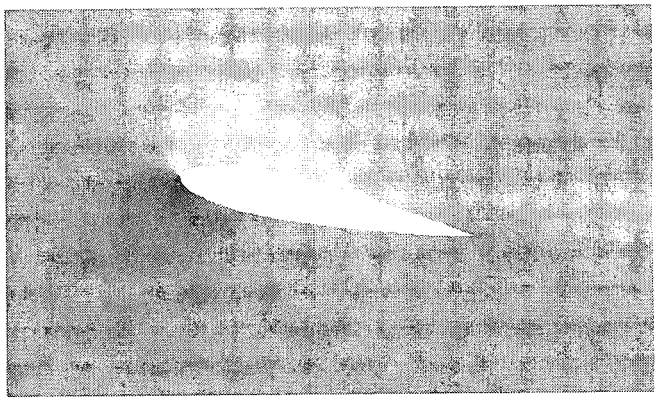

Figure 21: Subsonic Flow past a NACA0015, Square-Tip Wing in 3-Dimensions: pressure distribution at 95% span.

7.3 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds

A 0.50-cal. (1 cal. = 12.95 mm) spinning projectile is subjected to high speed, viscous flow at varying Mach numbers and 0 degree angle of attack. The computational model is 4.46 cal. in length with a 0.16-cal.-long, 0.02-cal.-deep groove and a 9 degree filleted boat tail. Figure 22 shows the projectile surface grid.

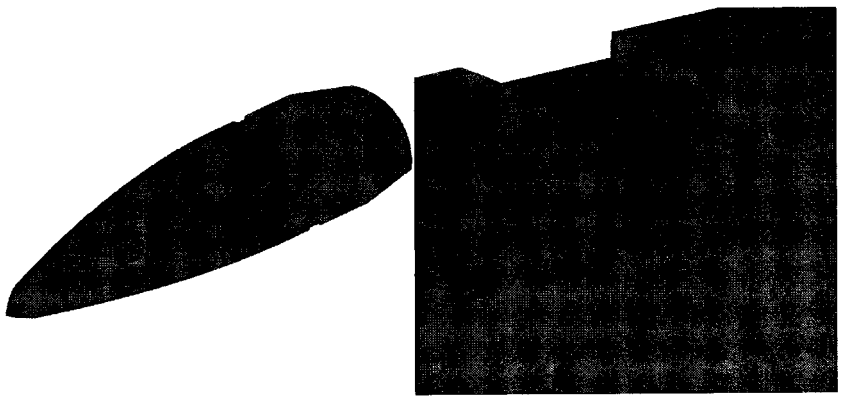

Figure 22: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: spinning projectile surface grid, and enlarged groove section view.

As can be seen from Figure 22, node clustering was done at regions of high curvature. A structured grid, Figure 23, consisting of 40,151,112 hexahedral elements was created.

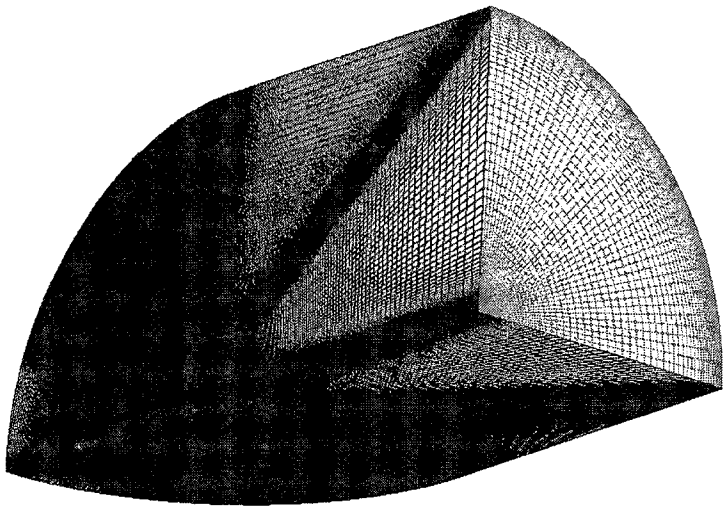

Figure 23: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: spinning projectile volume grid used in CFD calculations.

The outflow boundary is three projectile body lengths downstream of the projectile base, the inflow boundary is only half the body length upstream of the projectile nose, and the circumferential boundary is three body lengths away from the model. The mesh contained an initial boundary layer spacing of approximately 0.65 micrometers in height.

All experimental and numerical data used in this comparison were obtained from the U.S. Army Research Laboratory (ARL) [56] and covered a wide range of Mach numbers including subsonic, transonic, and supersonic flow regimes. Effects of 0 degree and 2 degree angles were investigated. A 2.7 million element hexahedral mesh was constructed for ARL simulations at supersonic speeds. The mesh contained a radial surface layer around the projectile body 1.07 micrometers in height.

The numerical results provided by the ARL were obtained using *CFD++* (Metacomp Technologies). *CFD++* solves the Reynolds-Averaged Navier-Stokes equations with

pointwise turbulence models. Both implicit and explicit time integration schemes are available. The point-implicit integration scheme was used to solve the steady state solution. Furthermore, the boundary layer is assumed to be fully turbulent and the pointwise k-ε turbulence model was utilized.

For all cases investigated, the free stream pressure and temperature are set to 101 kPa and 288 K, respectively. Using the perfect gas assumption, the resulting free stream density is 1.225 kg/m^3. The Reynolds number for all cases was determined using the projectile diameter, 1 cal. or 0.01295 m, as the characteristic length. For the projectile body, the boundary condition is set to be no slip, and rotating about the x-axis. The Mach numbers and corresponding roll rates are given in Table 1.

Table 1: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: Mach numbers and resulting roll rates used in CFD calculations.

Mach No.	Roll Rate (rad/s)
1.10	6172.5
1.25	7014.2
1.50	8417.0
2.00	11222.8
2.70	15150.6

Steady state computations were done on the Cray X1E. To accelerate the convergence toward steady solution, local time stepping is used. One nonlinear iteration is performed for the Navier-Stokes equations and one nonlinear iteration for the Spalart-Allmaras turbulence equation per time step. For Mach number cases 1.10 – 2.0, the Krylov space is set to 10 in the GMRES solver. 64 MSPs were used for these computations. Each computation achieved a minimum convergence of three orders of magnitude, which required approximately 8000 – 16,000 iterations. However, the Krylov space is set to 15 in the GMRES solver for Mach number case 2.7. Furthermore, this steady state case was

restarted from a previous unsteady calculation. The performance timings given below for this case reflect only a partial computation of 2000 time steps. The final computed drag coefficient, convergence order, total cpu time, and time-cost for each problem, according to Eq. (7.2), are given in Table 2.

Table 2: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: CFD calculation results and timings for Mach number investigations.

Mach No.	C_D	Time Steps	Convergence order	Time-Cost (Speed) (seconds)
1.10	0.376	16000	3.602836	1.915e-6
1.25	0.363	9000	3.249719	1.921e-6
1.50	0.335	8000	3.520705	1.921e-6
2.00	0.308	8000	3.571653	1.979e-6
2.70	0.254	2000	4.010311	1.880e-6

The following Figures, 24 - 28, depict the computed drag coefficient versus the time step for each problem and an image of the Mach number distribution corresponding to the final time step.

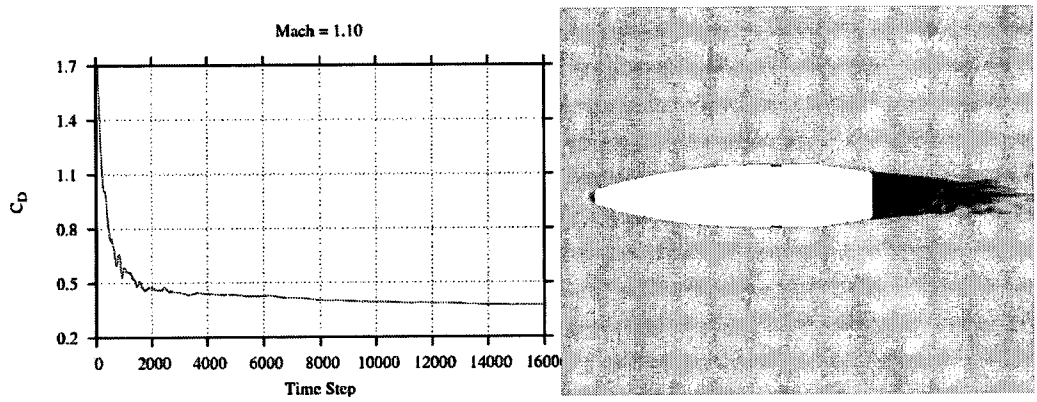

Figure 24: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: drag coefficient, left, and Mach distribution, right, at Mach 1.10.

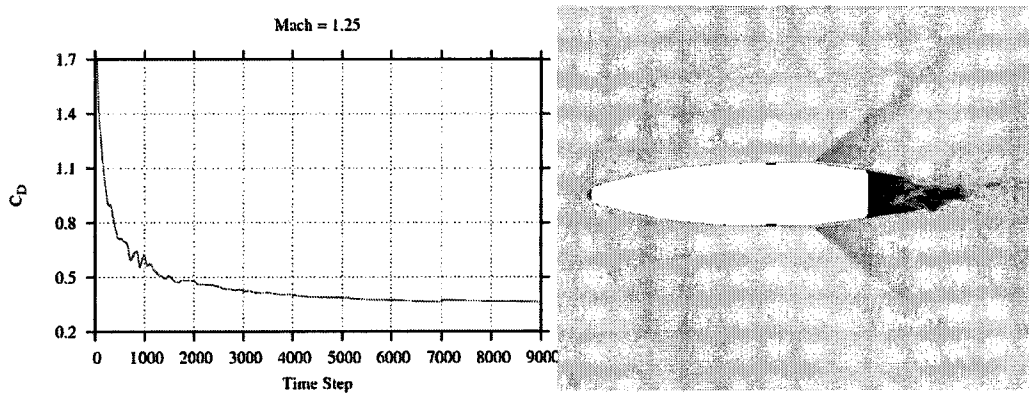

Figure 25: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: drag coefficient, left, and Mach distribution, right, at Mach 1.25.

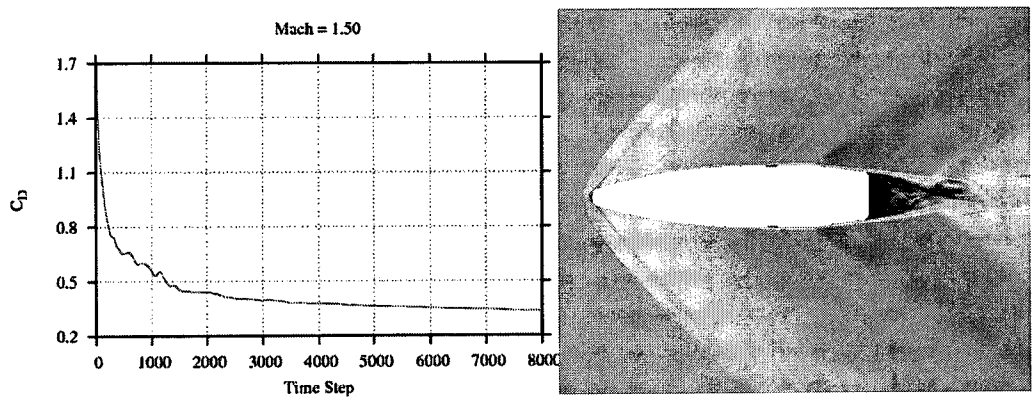

Figure 26: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: drag coefficient, left, and Mach distribution, right, at Mach 1.50.

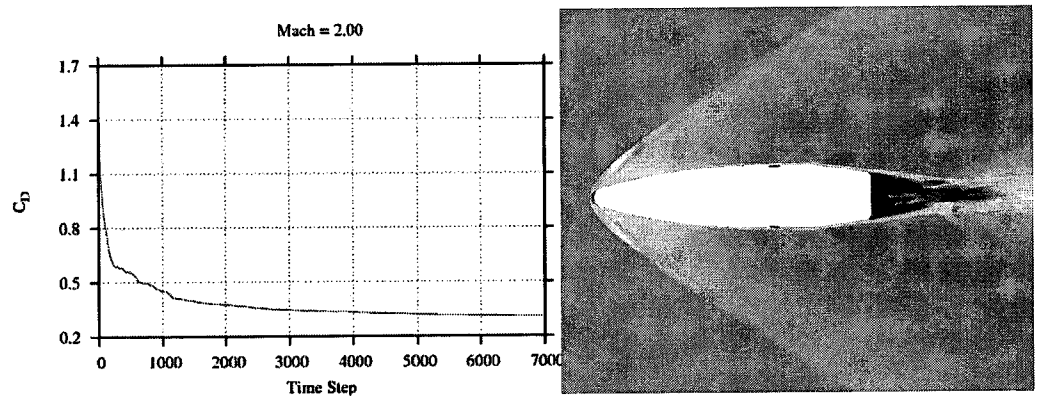

Figure 27: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: drag coefficient, left, and Mach distribution, right, at Mach 2.00.

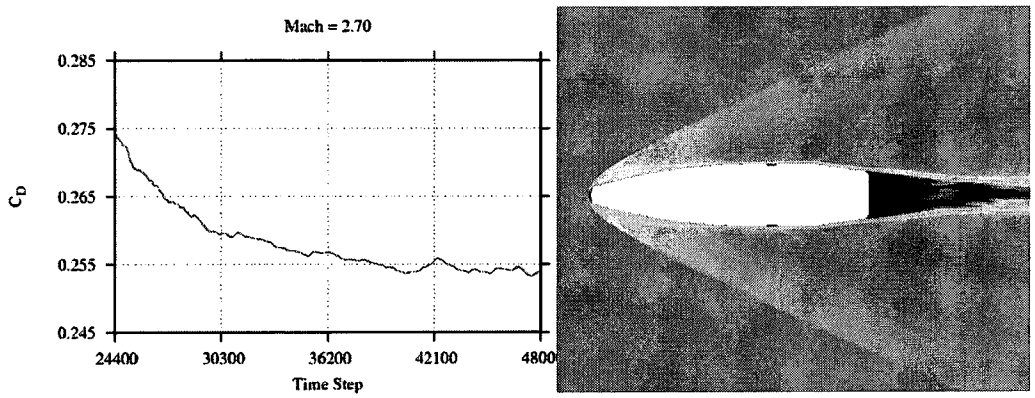

Figure 28: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: steady state drag coefficient, left, and Mach distribution, right, at Mach 2.70.

In all cases, the flow field exhibits a detached bow shock upstream of the projectile, which is expected for supersonic blunt body flows, followed by a region of flow with a velocity lower than that of the free stream velocity leading up to the stagnation point. Shocks also occur at the down stream edge of the groove section and again at the upstream edge of the filleted boat tail. The latter are a result of the flow expanding around the boat tail. At the base-edge, the boundary layer separates, resulting in a free shear layer, recirculating zone, and trailing wake. Due to the complexity of the interactions taking place in this region, it is often classified separately as base flow, which is briefly discussed later in this section.

Finally, the computed drag coefficients are plotted against the available ARL numerical and experimental results in Figure 29.

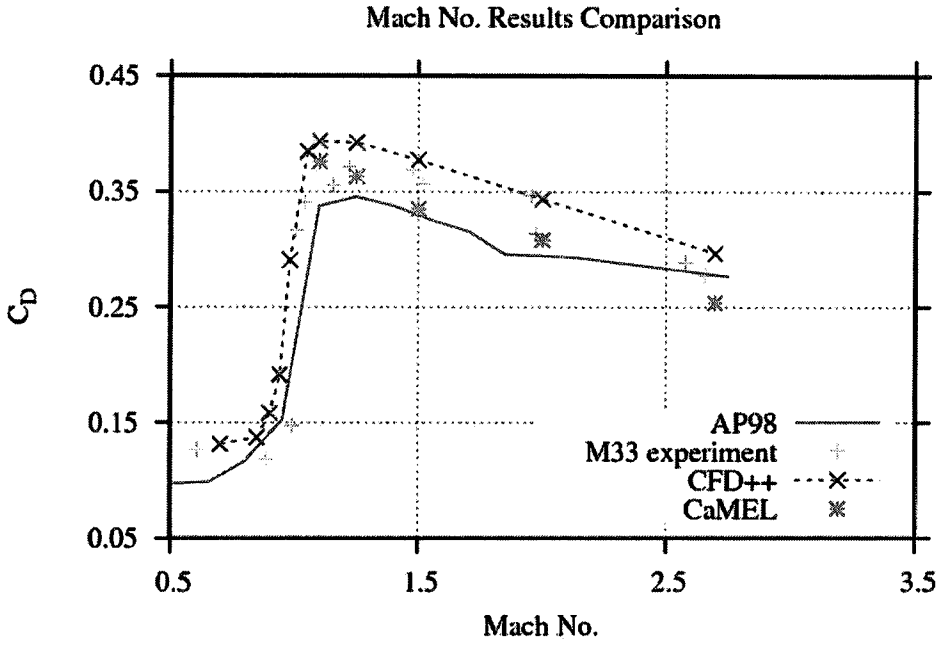

Figure 29: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: experimental and numerical results for investigated cases.

Refer to [56] for information about AP98 and M33 experimental results. The "CaMEL" data points refer to the results from our current finite volume solver. Keeping in mind that the geometry used to obtain the experimental data is not identical to the geometry used for numerical computations, the results obtained using our FV solver are in good agreement with the available comparison data.

To further illustrate the need for time accurate computations to capture the flow characteristics vital to accurate aerodynamic coefficient predictions, the Mach 2.70 was continued as an unsteady calculation for approximately 60 characteristic seconds. This time period is approximately 18 times the projectile body length, thus allowing for full development of the unsteady wake. Two nonlinear iterations are performed for the Navier-Stokes equations and one nonlinear iteration for the Spalart-Allmaras turbulence equation per time step, resulting in an average convergence of 2 orders of magnitude

within each time step. The Krylov space is set to 10 in the GMRES solver. 64 MSPs were used for this calculation. The time-cost of this application is approximately 1.85e-6 seconds. The computed drag coefficient is 0.25. Figure 30 displays the drag coefficient versus time and an image of the Mach number distribution for this case. Figure 31 depicts the steady state and unsteady wake for the Mach 2.70 case.

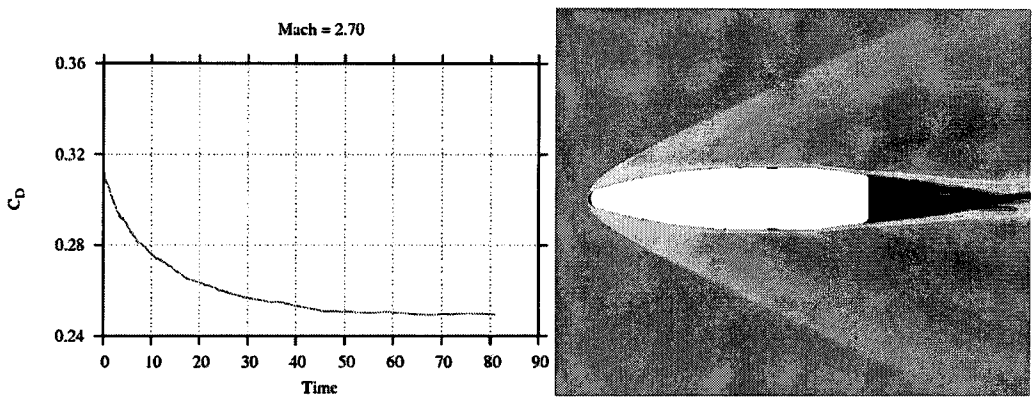

Figure 30: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: unsteady drag coefficient, left, and Mach distribution, right, at Mach 2.70.

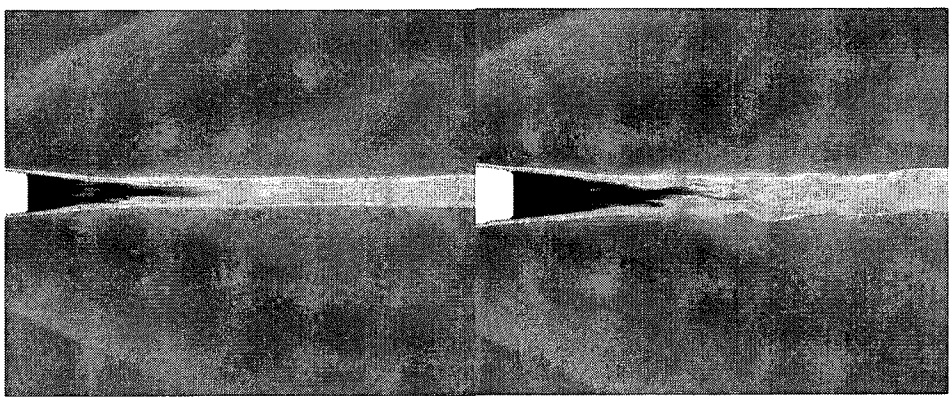

Figure 31: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: steady state wake, left, and unsteady wake, right, at Mach 2.70.

The difficulty involved in accurately predicting the aerodynamic coefficients for spinning projectiles, which are subject to Magnus forces, is well documented [57-61]. An added challenge for simulation of projectiles comes in the form of base flow

refers to a recirculation region located down stream of the projectile base-edge, and is the result of boundary layer separation. Within this region the pressure drops significantly lower than that of the free stream pressure. According to [62], for certain projectile configurations the base drag can contribute as much as 50% of the total drag. As such, it is vitally important that both grid resolution and the numerical scheme in use be finely tuned to capture this phenomenon. However, a clear understanding of base flow physics has yet to be established, due to the complicated interactions between the separated shear layers, recirculating zone, trailing wake, etc. Many researchers are currently investigated the characteristics of base flow [62-65].

Below, two Figures are presented. Figure 32 shows the drag coefficient due to pressure (pressure drag) and the drag coefficient due to friction (skin friction drag) for the unsteady Mach 2.70 case. In Figure 33, the drag coefficient for the projectile afterbody and base is shown. From Figure 33, it can be seen that the computed base drag contributed approximately 10% of the total drag.

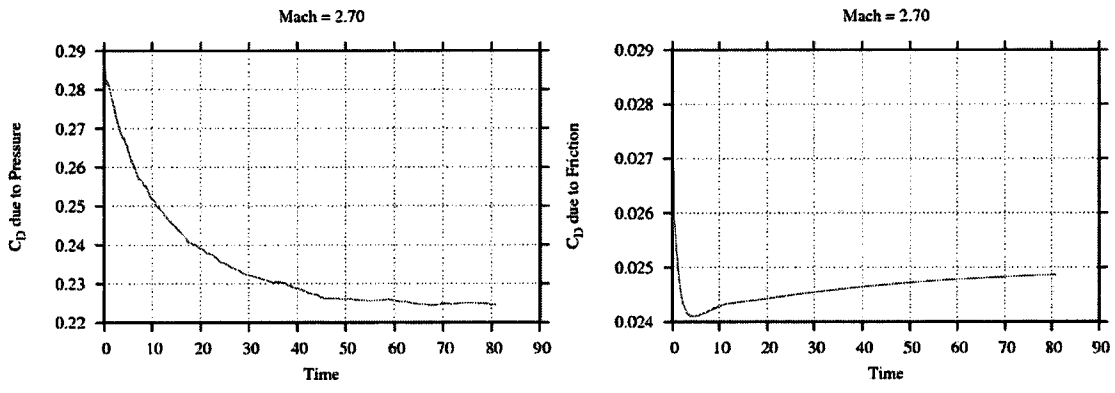

Figure 32: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: unsteady drag coefficient due to pressure, left, and unsteady drag coefficient due to friction, right, at Mach 2.70.

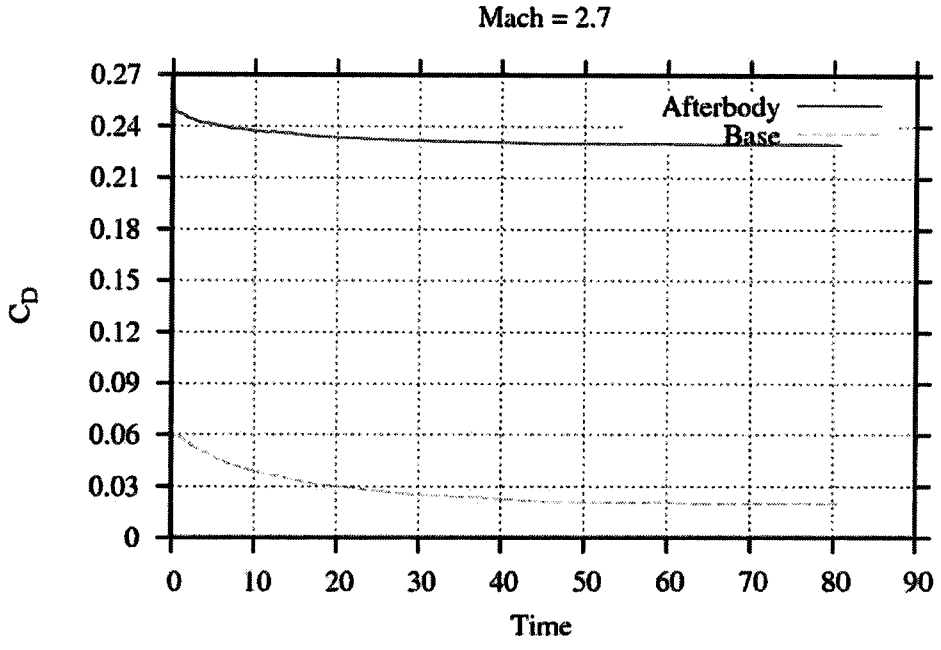

Figure 33: 3-Dimensional Flow past a Spinning Projectile at Supersonic Speeds: unsteady drag coefficient for projectile afterbody and base at Mach 2.70.

7.4 Code Scalability on Vector/Multi-Streaming and Linux Cluster Platforms

7.4.1 Linux Cluster Scalability

A test case of approximately 17.5 million tetrahedron cells is evaluated on the Linux cluster JVN located in Army Research Laboratory (ARL). Again, JVN contains 1024 nodes. Each node has dual 3.6 GHz Intel XEON EM64T processors and 4G memory with a peak system performance of 14.7 Teraflops. This case was run using 8, 32, and 64 processors for 100 time steps. The speedup vs. number of processors is shown in Figure 34.

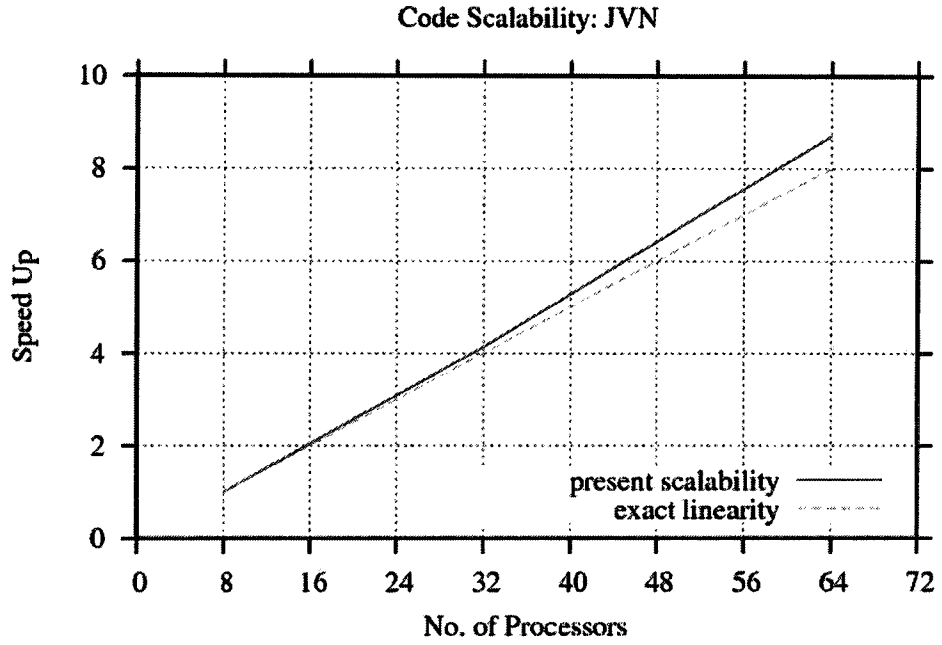

Figure 34: Code Scalability on Vector/Multi-Streaming and Linux Cluster Platforms: code scalability on the ARL JVN Linux cluster.

As can be seen, our solver exhibits super-linearity. This is due to the high performance communications network of JVN and the efficient communications subroutines written for the inter-processor communications using MPI functions. The physical timing for each case is presented in Table 3.

Table 3: Code Scalability on Vector/Multi-Streaming and Linux Cluster Platforms: physical timing reports for scalability investigations on JVN.

No. of Processors	Subtotal	Communication	Total	% Communication
8	55260.0	220.0	55480.0	0.40
32	13279.0	151.0	13430.0	1.12
64	6257.0	115.0	6372.0	1.80

7.4.2 Vector/Multi-Streaming Scalability

A test case of approximately 26.6 million prism/hexahedron cells is evaluated on the Cray X1E. This case was run using 32, 64, and 128 processors for 100 time steps. The speedup vs. number of processors is shown in Figure 35.

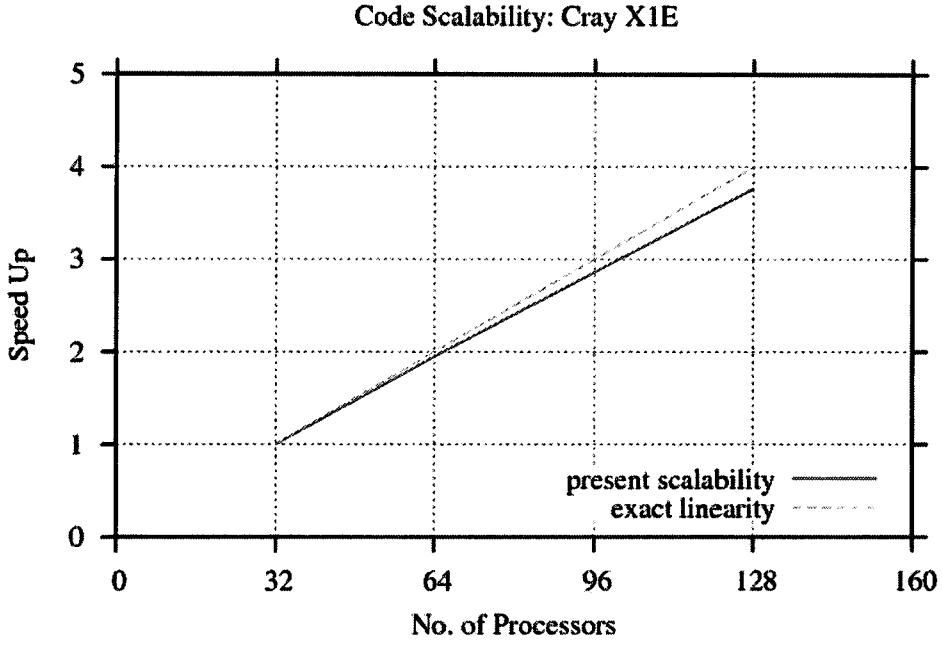

Figure 35: Code Scalability on Vector/Multi-Streaming and Linux Cluster Platforms: code scalability on the Cray X1E vector/multi-streaming supercomputer.

Again, our solver exhibits excellent scalability. The physical timing for each case is presented in Table 4.

Table 4: Code Scalability on Vector/Multi-Streaming and Linux Cluster Platforms: physical timing reports for scalability investigations on the Cray X1E.

No. of Processors	Subtotal	Communication	Total	% Communication
32	3254.2	65.6	3319.8	1.98
64	1661.7	42.1	1703.8	2.47
128	850.8	30.1	880.9	3.42

CHAPTER 8

CONCLUSIONS

In this dissertation, we have presented a matrix-free finite volume formulation for solving compressible flows on parallel, vector/multi-streaming, and linux cluster supercomputers.

We overviewed the compressible Navier-Stokes equations for viscous calculations. The semi-discrete formulation was used to simulate problems with fixed domains. We employ multiple Riemann solvers and linear reconstruction to ensure stability, robustness, and solution monotonicity in the advection dominated flow range. We addressed the implicit solution technique and also the Jacobian-free, matrix-free GMRES iterative algorithm used to solve the linear system of equations in the implicit computations. The LU-SGS preconditioner is used in conjunction with the GMRES method. Our matrix-free implementation drastically reduces the memory requirement for large-scale 3-dimensional applications.

We have implemented our finite volume formulation on the Cray X1E, a parallel, multi-streaming, vector processor supercomputer and the Linux Network Evolocity II cluster machine. We addressed the issues involved in vectorization of parallel algorithms and adopted a face-based data structure for more efficient solution computation and data scatter.

Finally we presented 2-dimensional and 3-dimensional examples which demonstrated the accuracy, applicability, and performance of the methods described. We have shown that our finite volume formulation is capable of solving problems at different Mach numbers, subsonic flows below the incompressible limit, and supersonic flows with strong shocks. In many of these test problems we have presented the robustness of the formulation at high Reynolds number flows. The applications considered in this dissertation have included the air flow past a spinning projectile, a NACA0015, square-tip wing, and a NACA0015, square-tip airfoil.

In summary, we have developed a formulation which, implemented on the Cray X1E or Linux cluster family of supercomputers, is capable of solving problems with the following features.

- Subsonic, transonic and supersonic external ballistics flows.
- Inviscid or viscous flows.
- Unstructured or structured meshes.
- Implicit methods.
- Jacobian-free GMRES solver.
- Matrix-free LU-SGS preconditioning.
- Hexahedral, tetrahedral, prism or pyramid elements.
- Linear Scalability.

Additionally, the typical speed, T_c, for a tetrahedral mesh simulation on the Cray X1E is approximately 1.0e-6 seconds for solving the Navier-Stokes equations and 3.4e-7 seconds for the turbulence equation. The speed for a hexahedral mesh simulation is approximately 1.3 - 2.0 times that of a tetrahedral mesh. The modular programming style used in

developing this finite volume solver makes it highly portable to other supercomputing platforms as well. In summary, we have developed a novel tool to aid Army and NASA researchers in the successful simulation and analysis of future aerodynamic platforms using state-of-the-art high performance computing platforms.

Finally, we would like to list some issues and recommendations which may improve the applicability of our finite volume solver.

- Many aerodynamic bodies have control surface for maneuverability and stability. The transient effects of the control surface motions in aerodynamic characteristics need to be studied. As such, incorporation of a mesh-moving scheme into the current solver for problems with moving boundaries would extend the applicability of our finite volume solver.
- Adaptive mesh refinement may improve the quality of our solutions. For problems with strong gradients, adaptively refining the mesh will greatly improve our solvers ability to represent these distinct flow features.

Future work will include additional scalability investigations of our solver on various supercomputing architectures. Specifically, this solver will be used in the TI-06 Capability Applications Project, in which several application codes will be given private access to high performance computing systems with 2,000 to 4,000 processors and three to nine terabytes of memory. Also, case studies for more complex geometries will be done using available experimental and thoroughly verified numerical data from government and industry research facilities, with emphasis in boundary layer separation and base flow.

REFERENCES

1. Anderson, J. D. Jr. *Fundamentals of Aerodynamics*. McGraw-Hill, New York, 2001.

2. Hughes, T. J. R. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, New York, 2000.

3. Bonet Chaple, R. P. Numerical Stabilization of Convection-Diffusion-Reaction Problems. *Delft Institute of Applied Mathematics (DIAM) Report*, 2006.

4. Tezduyar, T., and Sathe, S. Stabilization Parameters In SUPG AND PSPG Formulations. *Journal of Computational and Applied Mechanics*, Vol. 4, No. 1, pp.71-88, 2003.

5. Hughes, T. J. R., and Brooks, A. N. A Multidimensional Upwind Scheme with No Crosswind Diffusion. *Finite Element Methods for Convection Dominated Flows*, AMD Vol. 34, pp. 19-35. ASME, New York, 1979.

6. Hughes, T. J. R., Franca, L. P., and Hulbert, G. M. A New Finite Element Formulation for Computational Fluid Dynamics: VIII. The Galerkin/Least Square Method for Advective Diffusive Equations. *Computer Methods in Applied Mechanics and Engineering*, Vol. 73, pp. 173-189, 1989.

7. Oñate, E. A stabilized finite element method for incompressible viscous flow using a finite increment calculus formulation. *Computer Methods in Applied Mechanics and Engineering*, Vol. 182, No. 1-2, pp. 355-370, 2000.

8. Aliabadi, S., Abedi, J., Zellars, B. and Bota, K. New Finite Element Technique for Simulation of Wave-Object Interaction. *AIAA Paper 2002-0876*, 2002.

9. Aliabadi, S., Johnson, A., Abatan, A., Abedi, J., Yeboah, Y. and Bota, K. Stabilized Finite Element Formulation of Buoyancy Driven Incompressible Flows. *Journal of Communications in Numerical Methods in Engineering*, Vol. 18, Issue 5, pp. 315-324, 2002.

10. Abedi, J., and Aliabadi, S. Simulation of incompressible flows with heat and mass transfer using parallel finite element method. *Electronic Journal of Differential Equation*, Fifth Mississippi State Conference on Differential Equations and Computational Simulations, Conference 10, pp. 1-11, 2003.

11. Aliabadi, S., Shujaee, K., and Tezduyar, T. Parallel Simulation of Two-Phase Flow Problems Using the Finite Element Method. *Frontiers*, The 7th Symposium on the Frontiers of Massively Parallel Computation p. 113, 1999.

12. Aliabadi, S. *Parallel Finite Element Computations in Aerospace Applications.* PhD Thesis, University of Minnesota, 1994.

13. Aliabadi, S., and Tezduyar, T. Space-time finite element computation of compressible flows involving moving boundaries and interfaces. *Computer Methods in Applied Mechanics and Engineering* 107:209-223, 1993.

14. Aliabadi, S., and Tezduyar, T. Stabilized-finite-element/interface-capturing technique for parallel computation of unsteady flows with interfaces. *Computer Methods for Applied Mechanics and Engineering* 190:243-261, 2000.

15. Aliabadi, S., Abedi, J., Zellars, B., Bota, K., and Johnson, A. Simulation Technique for Wave Generation. *Communications in Numerical Methods in Engineering*, 19:349-359, 2003.

16. Aliabadi, S., Johnson, A., and Abedi, J. Comparison of Finite Element and Pendulum Models for Simulation of Sloshing. *Computer and Fluids* 32:535-545, 2003.

17. Aliabadi, S. Abedi, J. and Zellars, B. Parallel Finite Element Simulation of Mooring Forces on Floating Objects. *International Journal for Numerical Methods in Fluids* 41:809-822, 2003.

18. Aliabadi, S., Johnson, A., Abedi, J., Tu, S., and Tate, A. Simulation of contaminant dispersion on the Cray X1: verification and implementation. *Journal of Aerospace Computing, Information and Communication* 1:341-361, 2004.

19. Johnson, A. Computational fluid dynamics applications on the Cray X1 architecture: experiences, algorithms, and performance analysis. *Cray User Group Conference 2003 Proceedings*, 2003.

20. Barth, T., and Ohlberger, M. Finite volume methods: foundation and analysis. *Encyclopedia of Computational Mechanics*, John Wiley & Sons, Ltd., 2004.

21. Toro, E. *Riemann solvers and numerical methods for fluid dynamics.* Springer, New York, 1999.

22. Sun, M., and Takayama, K. An artificially upstream flux vector splitting scheme for the Euler equations. *Journal of Computational Physics* 189:305-329, 2003.

23. Dumbser, M., Moschetta, J.-M., and Gressier, J. A Matrix Stability Analysis of the Carbuncle Phenomenon. *Journal of Computational Physics*, Vol. 197, Issue 2, pp. 647-670, 2004.

24. Robinet, J.-Ch., Gressier, G. C., and Moschetta, J.-M. Shock wave instability and carbuncle phenomenon: same intrinsic origin? *Journal of Fluid Mechanics*, 417:237-263, 2000.

25. Harten, A., Lax, P., and van Leer, B. On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. *SIAM Review* 25:35-61, 1983.

26. Batten, P., Clarke, N., Lambert, C., and Causon, D. M. On the Choice of Wavespeeds for the HLLC Riemann Solver. *SIAM Journal on Scientific Computing*, Vol. 18, No. 6, pp. 1553-1570, 1997.

27. Einfeldt, B., Munz, C. D., Roe, P. L., and Sjogreen, B. On Godunov-type Methods Near Low Densities. *Journal of Computational Physics*, 92:273-295, 1991.

28. Scalabrin, L. C., and Boyd, I. D. Development of an unstructured Navier-Stokes solver for hyperbolic nonequilibrium aerothermodynamics. *AIAA 2005-5203*, 2005.

29. Donat, R., and Marquina, A. Capturing Shock Reflections: An Improved Flux Formula. *Journal of Computational Physics*, 25:42-58, 1996.

30. Toth, G., and Odstrcil, D. Comparison of some Flux Corrected Transport and Total Variation Diminishing Numerical Schemes for Hydrodynamic and Magnetohydrodynamic Problems. *Journal of Computational Physics*, 128:82, 1996.

31. Roe, P. L. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43:357-372, 1981.

32. Berger, M., and Aftosmis, M. J. Analysis of Slope Limiters on Irregular Grids. *AIAA 2005-0490*, 2005.

33. Doering, C. R., and Gibbon, J. D. *Applied Analysis of the Navier-Stokes Equations*. Cambridge University Press, 1995.

34. Tejada-Martinez, A. E. *Dynamic Subgrid-Scale Modeling For large-Eddy Simulation Of Turbulent Flows With A Stabilized Finite Element Method.* PhD Thesis, Rensselaer Polytechnic Institute, 2002.

35. Yan, H., and Knight, D. Large Eddy Simulation of Supersonic Flat Plate Boundary Layer Part II. *AIAA 2002-4286*, 2002.

36. von Kaenel, R. *Large-Eddy Simulation Of Compressible Flows Using The Finite Volume Method.* PhD Thesis, Swiss Federal Institute of Technology, 2003.

37. Lien, F. S., Kalitzin, G., and Durbin, P. A. RANS modeling for compressible and transitional flows. *Center for Turbulence Research Proceedings of the Summer Program*, 1998.

38. Nikitin, N.V., Nicoud, F., Wasistho, B., Squires, K.D., and Spalart, P.R. An approach to wall modeling in large-eddy simulations. *Physics of Fluids* 12:1629-1632, 2000.

39. Catris, S., and Aupoix, B. Density corrections for turbulence models. *Aerosp. Sci. Technol.* 4:1-11, 2000.

40. Forsythe, J. R., Hoffman, K. A., Cummings, R. M., and Squires, K. D. Detached Eddy Simulation With Compressibility Corrections Applied to a Supersonic Axisymmetric Base Flow. *Journal of Fluids Engineering* 124:911-923, 2002.

41. Constantinescu, G. S., and Squires, K. D. LES and DES Investigations of Turbulent Flow over a Sphere at Re = 10,000. *Flow, Turbulence and Combustion* 70:267-298, 2003.

42. Kapadia, S., and Subrata, R. Detached Eddy Simulation Over a Reference Ahmed Car Model. *AIAA 2003-0857*, 2003.

43. Petit, G., and Thompson, S. R. Performance Analysis of the ARL Linux Network Cluster. *Proceedings of the Users Group Conference (DOD_UGC'04)*, 2004.

44. Saad, Y. *Iterative methods for sparse linear systems*. PWS Publishing Company, 1996.

45. Knoll, D., and Keyes, D. Jacobian-free Newton-Krylov methods: a survey of approaches and applications. *Journal of Computational Physics* 193:357-397, 2004.

46. Luo, H., Baum, J., and Lohner, R. A fast matrix-free implicit method for compressible flows on unstructured grids. *Journal of Computational Physics* 146:664-690, 1998.

47. Sharov, D., Luo, H., Baum, J., and Lohner, R. Implementation of unstructured grid GMRES+LU-SGS method on shared-memory cache-based parallel computers. *AIAA 2000-0927*, 2000.

48. Tu, S., Aliabadi, S., Johnson, A., and Watts, M. High performance computation of compressible flows on the Cray X1. *Proceedings of the Second International Conference on Computational Ballistics*, Cordoba, Spain 347-356, 2005.

49. Benzi, M., Tuma, M. A comparative study of sparse approximate inverse preconditioner. *Applied Numerical Mathematics* 30:305-340, 1999.

50. Tu, S., Aliabadi, S., Johnson, A., and Watts, M. A robust parallel implicit finite volume solver for high-speed compressible flows. *AIAA 2005-1396*, 2005.

51. Tu, S., and Aliabadi, S. A slope limiting procedure in discontinuous Galerkin finite element method for gasdynamics applications. *International Journal of Numerical Analysis and Modeling* 2:163-178, 2005.

52. Mathur, S. R., and J.Y. Murphy, J. Y. A pressure-based method for unstructured meshes. *Numerical Heat Transfer, Part B* 31:195-215, 1997.

53. Hyams, D. G., Sreenivas, K., Sheng, C., Briley, W. R., Marcum, D. L., Whitfield, D. L. An investigation of parallel implicit solution algorithms for incompressible flows on multielement unstructured topologies. *AIAA 2000-0271*, 2000.

54. Karypis, G., and Kumar, V. *MeTiS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-reducing Orderings of Sparse Matrices, Version 4.0*. Dept. of Computer Science and Engineering, University of Minnesota, 1998.

55. Lin, C. -W., Percival, W. S., Gotimer, E. H. Viscous drag calculations for ship hull geometry. *Technical report*, Carderock Division Naval Surface Warfare Center, Bethesda, Maryland, 1995.

56. Silton, S. I. Navier-Stokes Computations For A Spinning Projectile From Subsonic To Supersonic Speeds. *AIAA 2003-3936*, 2003.

57. Dupuis, A., Bernier, A., and Hathaway, W. Magnus Instabilities And Modeling For A 12.7 mm Projectile With Roll Damping Finlets From Free-Flight Tests At Mach 1.7. *19th International Symposium of Ballistics*, Interlaken, Switzerland, 7-11 May, 2001.

58. DeSpirito, J., Edge, L., Vaughn, M. E. Jr., and Washington, W. D. CFD Investigation of Canard-Controlled Missiles with Planar and Grid Fins In Supersonic Flow, *AIAA 2002-4509*, 2002.

59. Sahu, J., Edge, H., Dinavahi, S., and Soni, B. Progress on Unsteady Aerodynamics of Maneuvering Munitions. *Users Group Meeting Proceedings*, Albuquerque, NM, June 2000.

60. Sahu, J. High Performance Computing Simulations for a Subsonic Projectile with Jet Interactions. *Proceedings of the Advanced Simulations and Technologies Conference*, San Diego, CA, April 2002.

61. Sahu, J. Time-Accurate Numerical Prediction of Free Flight Aerodynamics of a Finned Projectile. *AIAA 2005-5817*, 2005.

62. Kawai, S. *Computational Analysis of the Characteristics of High Speed Base Flows*. Ph.D. Thesis, University of Tokyo, 2005.

63. Subbareddy, P., and Candler, G. Numerical Investigations of Supersonic Base Flows Using DES. *43rd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 10-13 Jan, 2005.

64. Sahu, J., and Nietubicz, C. Navier-Stokes Computations of Projectile Base Flow with and without Mass Injection. *AIAA Journal*, Vol. 23, No. 9:1348-1355, 1985.

65. Herrin, J. L., and Dutton, J. C. Supersonic Base Flow Experiments in the Near-Wake of a Cylindrical Afterbody. *AIAA Journal*, Vol. 32, No. 1:77-83, 1994.

66. Kennel, M. B. *KDTREE2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space*. http://arxiv.org/PS_cache/physics/pdf/0408/0408067.pdf, 2004.